HD/ backup
- DRVR
- FORMAT
- ndskopen
- MOUNT
- mounttable
- name
- OPEN
- probootdrvr
- profilebootblks
- rsrc
- source

wdgtbootdrvr
wdgtbootblks
writemacbblks
wrtbblks
make/HDinstall

```
{*****************************************************************************}
{start of main body of HDI (Hard Disk Install)}

BEGIN
MainInit;
InitBuffer;                         { set up the copy buffer }
InitMyDialogs;                      { set up all the dialog pointers }

SetCursor(watch);
AppRefID := CurResFile;                 { save ptr to app's resource file }
result := HDOpen;                       { Open the driver, add drives to drive queue, try mount }

{check for fatal errors}
SetCursor(arrow);
CheckErr(result);

IF (result = MWorksFnd) THEN
   BEGIN                                { if direct-start disk, ask how to proceed }    {19Apr85}
   LDilog := GetNewDialog(MWDilog, NIL, POINTER(-1));                                   {19Apr85}
   ModalDialog(NIL, itemhit);                                                           {19Apr85}
   DisposDialog(LDilog);                { remove dialog }                               {19Apr85}

   CASE itemhit OF                                                                      {19Apr85}
      1:                                { Initialize button pressed }                   {19Apr85}
         BEGIN                                                                          {19Apr85}
         DoInstall(ErasDilog, result, BlankDisk);   { erase entire drive }              {19Apr85}
         IF (result = UserCancel) THEN                                                  {19Apr85}
            ByeBye;                                                                     {19Apr85}
         END;                                                                           {19Apr85}

      2:                                { Replace MacWorks button pressed }             {19Apr85}
         BEGIN                                                                          {19Apr85}
          DoMWInstall;                                                                  {19Apr85}
         END;                                                                           {19Apr85}

      3:                                { Cancel button pressed }                       {19Apr85}
         ByeBye;                                                                        {19Apr85}
      END; {of CASE}                                                                    {19Apr85}
   END {of IF THEN}                                                                     {19Apr85}

ELSE IF (result = LisaFnd) THEN
   BEGIN                                { Lisa OS format, ask about sharing }
   LDilog := GetNewDialog(LisaDilog, NIL, POINTER(-1));
   ModalDialog(NIL, itemhit);
   DisposDialog(LDilog);                { remove dialog }

   CASE itemhit OF
      1:                                { Share button pressed }
         BEGIN
         result := HDMount;             { Try to do mount of drive }
         CheckErr(result);              { Check for fatal errors }
         IF (result = AllOK) THEN       { Mac side OK, say "already a shared disk"... }
            DoInstall(SharDilog, result, Lisadisk)
         ELSE IF (result < 0) THEN      { Mac side bad, say "unreadable, do you want to format".. }
            BEGIN
            DoInstall(BadSDilog,result,Lisadisk);
```

```
            IF (result = UserCancel) THEN
               ByeBye
            END;
         END;

      2:                                  { Erase All button pressed }
         BEGIN
         DoInstall(ErasDilog, result, BlankDisk);   { erase entire drive }
         IF (result = UserCancel) THEN
            ByeBye;
         END;

      3:                                  { Cancel button pressed }
         ByeBye;
      END; (of CASE)
   END (of IF THEN)

ELSE IF (result = AllOK) THEN              { Mac disk on-line }
   DOInstall(MacDilog, result, MacDisk)    { check if reinitialize wanted }

ELSE                                       { damaged or not initialized }
   BEGIN
   DoInstall(BadMDilog, result, BlankDisk);    { erase entire drive }
   IF (result = UserCancel) THEN
      ByeBye;
   END;

CASE (result) OF

   AllOK:
      BEGIN
      result := WrtBootBlks;                  { write boot blocks to hard disk }
      IsIconInSysRes;                         { make sure hard disk icons are in sysres }
      SetCursor(arrow);
      itemhit := NoteAlert(LstAlert, NIL);    { notify about moving system folder }
      END;

   MWorksFnd:                                 { MacWorks has been replaced - all done }    {19Apr85}
      BEGIN                                                                                 {19Apr85}
      END;                                                                                  {19Apr85}

   UserCancel:
      IsIconInSysRes;                         { make sure disk icons are in sysres }

   OTHERWISE
      BEGIN
      Cvtnum(result, resstrng);               { convert error to string }
      paramtext(resstrng, '', '', '');        { set error id for alert display}
      itemhit := StopAlert(InitAlert, NIL)    { Display error }
      END;
   END; (of CASE)

99:                                        { bail out point}
HideCursor;                                { return to the Finder . . .}

END.
```

```
PROGRAM HDI;     { HardDisk install program -- from Rich Castro's Hard Disk Install & MacWorks Install }

{ **** MacWorks Install documentation ****)

{ Mac application for installing MacWorks to a hard disk.  The steps are as follows.
  1) Eject application disk and ask for MacWorks diskette.
  2) Verify inserted disk as MacWorks and read into buffer.
  3) Copy 400K to first 800 blocks of hard disk (actually offset by 8 blocks).
  4) Copy Monitor Profile or Widget bootblocks from MWI/68K to first 8 blocks on hard disk.
  5) Set the hard disk driver offset for MacWorks (= 810)
  6) Quit }

{ **** List-O-Minus-O-Matic-O-Things-To-Do **** }

{ **** Modification history (MWInstall) **** }
{ 25-Oct-84    New Today from MacWorks Copy application }
{ 26-Oct-84    Resource editing, Profile bootblk = 4K }
{ 29-Oct-84    More editing, tag buffer setup, etc }
{ 30-Oct-84    New control calls, error fixing }
{ 31-Oct-84    Cosmetic Hello/Goodbye dialogs }
{  5-Nov-84    Last dialog display fixup, Widget bootblock setup }
{  6-Nov-84    Terminates if running off of the hard disk, more Widget stuff, unmounts hard
               disk at end of process }
{  7-Nov-84    Unmount before writing to hard disk, as unmount flushes, needs directory }
{  7-Nov-84    Multiple fixes to port setting, also dialog fix }
{  8-Nov-84    Dispose of generic dialog before adios, force sony on-line before end }

{ **** Hard Disk Install documentation **** }

{ Application routine to enable initial install of Hard disk driver and    }
{ mount/initialize of hard disks for Lisa system running MacWorks.         }
{                                                                          }
{ Written by Rich Castro - 3/9/84                                          }
{                                                                          }

{ **** Modification history (HDInstall) **** }
{    3/29/84 - add support for new icon, cursor changes                    }
{    4/2/84  - change icn# id for system id                                }
{    4/9/84  - add support for init device error                           }
{    5/3/84  - add support for Lisa/Mac disk sharing                       }
{    5/8/84  - add writing of icon to system resource file                 }
{    5/10/84 - add display of error code for open errors                   }
{    5/11/84 - add ability to reinitialize a Mac disk                      }
{    5/14/84 - add alert for moving system folder                          }
{    5/16/84 - add disk type parameter to HDInstall interface              }
{    5/25/84 - add internal hard disk icon                                 }
{    6/1/84  - remove dialog about auto-install - always do it             }
{    6/4/84  - add new dialogs                                             }
{    6/21/84 - change hard disk id to 1, change icon id's also             }
{    6/29/84 - add write boot blocks kluge if install successful           }
{    7/18/84 - eliminate separate dialog file                              }

{ **** Joint Modification History **** }

{  9-Nov-84    If doing MWInstall, set error before call to HDFormat }
{ 12-Nov-84    Alert/dialog cleanup }
```

```
{ 17-Nov-84    Cancel out of MWInstall, format before MWInstall, HDName added, cleanup }
{ 18-Nov-84    Set res file back to application after IsIconInSysRes call }
{ 20-Nov-84    Passes disk type to HDFormat (again) }
{ 12-Dec-84    Don't coerce UserCancel err to AllOK, add UserCancel end case handling }
{ 19-Apr-85    Add option to update MacWorks only}

{ **** Compiler directives **** }
{$OV-}   { no integer overflow checking }
{$M+}    { auto Mac-style code generation }
{$R-}    { range checking off }
{$X-}    { no automatic stack expansion code }
{$D-}    { symbols off }
{$U-}    { turn off Lisa libraries }

USES
     {$U -priam-mac/obj-MemTypes    } MemTypes,
     {$U -priam-mac/obj-QuickDraw   } QuickDraw,
     {$U -priam-mac/obj-OsIntf      } OsIntf,
     {$U -priam-mac/obj-toolIntf    } ToolIntf,
     {$U -priam-mac/obj-PackIntf    } PackIntf;

LABEL
   99;      {for bailing out of program}

CONST

   { **** MacWorks Install Constants **** }

   { 'DLOG' rsrc ID's }

   MyDlgID     = 281;         { generic dialog template, one string & one button }

   { Equates for Hello dialog responses }

   {Install    = 1;
   {Cancel     = 2;

   { Equates for Adios dialog reponse }

   {OK         = 1;

   { 'STR ' rsrc ID's for the text inside the button }

   OKbtnID     = 281;
   CancelbtnID = 282;

   { 'STR#' rsrc ID for the string list containing all the dialog text }

   MyStrID     = 281;

   {Location in strHandle for strings, from ID's passed to DisplayMsg.
    The disk dialogs prompt the user for a disk and have a cancel button}

   GetOldMW    = 1;        {'insert source MacWorks diskette'}

   {the wait dialogs tell the user something's in progress}
```

```
DoingRead   = 2;
DoingWrite  = 3;

{The stop dialogs say something couldn't be done, with an OK button}
CantWrite   = 4;
CantEject   = 5;
NotMWDisk   = 6;
CantInit    = 7;
CantRead    = 8;
CantControl = 9;

{ Control call values }

SetTagCode     = 8;     { Set tag buffer ptr control call value }
SetOffsetCode  = 9;     { Set first mac block offset value }
InitDrvrCode   = 30;    { Initialize the hard disk driver }

{ Miscellaneous equates }

StatusCode     = 8;     { Value of disk driver status request param }
NumBootBlks    = 8;     { Number of boot blocks to write out }
LastMonBlk     = 810;   { Number of blocks on hard disk reserved for MacWorks }
HDdrvrRefnum   = -2;    { Hard disk driver ID = 1 (refnum = NOT 1 = -2) }
SonyDrvrRefnum = -5;    { Sony disk driver ID = 4 (refnum = NOT 4 = -5) }
HDVRefnum      = 4;     { Hard disk driver default volume number (paraport) }
SonyVRefnum    = 1;     { Sony disk driver volume number (internal drive) }

{ **** Hard Disk Install Constants **** }

AllOK       = 0;            { No error }
UserCancel  = 1;            { User cancelled out of an operation }
OpenErr     = 2;            { Error opening the hard disk driver??? }
ResrcErr    = 3;            { Error getting/setting resource }
NoDrvErr    = 4;            { No hard disk attached }
NtLisaErr   = 5;            { User running program on a Mac }
NoSpace     = 6;            { Disk was Lisa OS format, but not partitioned }
NtRevCErr   = 7;            { Lisa not booted from Rev C or later MacWorks }
OnHDiskErr  = 8;            { User running program off of the hard disk }
MWdiskErr   = 9;            { Bogus error for call to HDFormat }

OnlineErr  = -55;
MWorksFnd  = -91;            { Macworks hard disk attached }
TimeErr    = -95;
RespErr    = -96;
PrtyErr    = -97;
HardErr    = -98;
LisaFnd    = -99;            { Lisa OS hard disk attached }

NtLisaAlert = 128;
InitAlert   = 129;
NoDskAlert  = 130;
LstAlert    = 132;
SpcAlert    = 133;
OnHDiskAlert = 134;
NtRevCAlert = 135;
MWnoteAlert = 136;
```

```
    MacDilog   = 140;
    BadMDilog  = 141;
    NameDilog  = 142;
    InitDilog  = 143;
    LisaDilog  = 145;
    SharDilog  = 146;
    BadSDilog  = 147;
    ErasDilog  = 148;
    MWDilog    = 149;

    InitID     = 4;
    DrvrID     = 1;
    ProIcon    = -16351;
    IntrnlIcon = -16350;

    {NOTE:  The only true importance of the following 3 equates is that they are passed to HDFormat,
     the assembly routine that formats (initializes) the hard disk.  HDFormat passes the parameter
     down one more level to the hard disk driver.  If the parameter = 0, the disk driver resets to
     volume size to be the total size of the volume (sets the logical to physical offset to 0),
     otherwise the offset is left alone;  the disk is then initialized from block 0 (logical) to
     the last block of the volume.  Since the only two types of disk that we support are 'Shared'
     and MacWorks only (disk partitioned between MacWorks (810 blks) and Macintosh), we would only
     zero the entire disk if the disk was going from damaged/unformatted/Lisa format to MacWorks
     only (macintosh disk w/MacWorks image in first 810 blocks).  The others all test
     to non-zero by the driver, thus leaving the offset unchanged (which has been previously set
     up by the call to HDOpen, which called the initialization routine of the hard disk driver).
     The offset could also be set prior to writing out the directory by DoMWInstall}

    BlankDisk  = 0;      { initialize the entire disk as a Macintosh volume }
    LisaDisk   = 1;      { initialize logical disk, skip part formatted for Lisa OS }
    MacDisk    = 2;      { initialize logical disk, skip part formatted for MacWorks }

    WatchID    = 4;      {system ID for watch cursor}

TYPE
    TdiskMode   = (FirstHalf, LastHalf, BothHalf, TwoBlks);  { data read/write modes }
    TmsgType    = (WaitMsg, StopMsg, DiskMsg);               { different flavors of my dialogs }

VAR
    { **** MacWorks Install Variables **** }

    bigGuy      : BOOLEAN;        { TRUE => 1Meg Lisa, 400K copy buffer, do it in one pass }
    bootBlkPtr  : Ptr;            { Pointer to boot block code }
    bootDiskOut : BOOLEAN;        { TRUE => Boot sony ejected }
    btnCancel   : StringHandle;   { Handle to 'Cancel' button text }
    btnOK       : StringHandle;   { Handle to 'OK' button text }
    copyBuffer  : Ptr;            { Pointer to copy buffer }
    firstDialog : BOOLEAN;        { TRUE => show the dialog window and draw it in DisplayMsg }
    itemHit     : INTEGER;        { generic dialog response }
    myBtnHdl    : ControlHandle;  { 'cancel' button control handle }
    myDlgPtr    : DialogPtr;      { generic dialog pointer }
    myHDPBlk    : ParamBlockRec;  { hard disk I/O parameter block }
    myHDPBptr   : ParmBlkPtr;     { ptr to hard disk I/O parameter block }
    myMDPBlk    : ParamBlockRec;  { micro disk I/O parameter block }
    myMDPBptr   : ParmBlkPtr;     { ptr to micro disk I/O parameter block }
```

```
    myTextHdl   : Handle;        { handle to statText in dialog box }
    statusPBptr : ParmBlkPtr;    { for status calls }
    tempDlgPtr  : DialogPtr;     { Hello dialog ptr }
    watch       : Cursor;

    { **** Hard Disk Install Variables **** }

     appRefID      : INTEGER;
     result        : INTEGER;
     LDilog        : DialogPtr;
     resstrng      : Str255;

{*****************************************************************************}

PROCEDURE MacsBug;                              INLINE $A9FF;
PROCEDURE MacsBugPrint (theMsg: str255);        INLINE $ABFF;

{ **** MacWorks Install External Procedures ****}

PROCEDURE ProfileBoot;                          EXTERNAL;   { start of Profile boot code }
PROCEDURE WidgetBoot;                           EXTERNAL;   { start of Widget boot code }

{ **** Hard Disk Install External Procedures ****}

FUNCTION HDOpen : INTEGER;                        EXTERNAL;
FUNCTION HDMount : INTEGER;                       EXTERNAL;
FUNCTION HDFormat (diskType: INTEGER) : INTEGER;  EXTERNAL;
FUNCTION HDName : INTEGER;                        EXTERNAL;
FUNCTION WrtBootBlks : INTEGER;                   EXTERNAL;

{*****************************************************************************}
{forward declarations of all procedures}

{ **** MacWorks Install Procedures **** }

FUNCTION  ClickedBtn (VAR theEvent: EventRecord) : BOOLEAN;          FORWARD;
PROCEDURE DisplayMsg (theMsg: INTEGER; theType: TmsgType; doExit: BOOLEAN);
                                                                     FORWARD;
PROCEDURE DoMWInstall;                                               FORWARD;
PROCEDURE EjectDiskette;                                             FORWARD;
PROCEDURE ForceOffline;                                              FORWARD;
PROCEDURE ForceRemount;                                              FORWARD;
PROCEDURE GetMWDisk;                                                 FORWARD;
PROCEDURE HDControl (theCode: INTEGER; theValue: Ptr; abort: BOOLEAN); FORWARD;
PROCEDURE HDStatus (VAR theStatus: ParmBlkPtr);                      FORWARD;
PROCEDURE InitBuffer;                                                FORWARD;
PROCEDURE InitMyDialogs;                                             FORWARD;
PROCEDURE MainInit;                                                  FORWARD;
FUNCTION  MWDisk : BOOLEAN;                                          FORWARD;
PROCEDURE ReadBootBlks;                                              FORWARD;
PROCEDURE ReadDiskette (readMode: TdiskMode);                        FORWARD;
PROCEDURE Terminate (why: INTEGER);                                  FORWARD;
PROCEDURE WriteBootBlocks;                                           FORWARD;
PROCEDURE WriteToDisk (writeMode: TdiskMode);                        FORWARD;

{ **** Hard Disk Install Procedures **** }
```

```
PROCEDURE ByeBye;                                                    FORWARD;
PROCEDURE CheckErr (error: INTEGER);                                 FORWARD;
PROCEDURE CvtNum (num: INTEGER; VAR str: str255);                    FORWARD;
PROCEDURE DoInstall (DilogID: INTEGER; VAR error: INTEGER; drivetype: INTEGER);
                                                                     FORWARD;
PROCEDURE IsIconInSysRes;                                            FORWARD;

{ **** Start of MacWorks Install Procedures **** }

{*****************************************************************************}
FUNCTION  ClickedBtn (VAR theEvent: EventRecord) : BOOLEAN;
{called when there's a mouse-down event and we have a 'insert a disk' dialog box
 up.  Returns TRUE only if user really selects the 'Cancel' button in the box}

VAR
   i           : INTEGER;
   myPort      : GrafPtr;
   savePort    : GrafPtr;
   tempCtrl    : ControlHandle;
   tempWindow  : WindowPtr;

BEGIN
ClickedBtn := FALSE;

i := FindWindow(theEvent.where, tempWindow);       { which window was the mouse down in? }
IF (tempWindow = WindowPtr(myDlgPtr)) THEN
   BEGIN {in the dialog window, check for button hit}
   GetPort(savePort);
   SetPort(myDlgPtr);                              { make my dialog the window for globtoloc conversion }
   GlobalToLocal(theEvent.where);
   i := FindControl(theEvent.where, tempWindow, tempCtrl);
   IF (tempCtrl = myBtnHdl) THEN                   { user mouse-downed in the 'Cancel' button }
      IF (TrackControl(tempCtrl, theEvent.where, NIL) <> 0) THEN  {user selected cancel}
         ClickedBtn := TRUE;
   END
ELSE
   SysBeep(10);

SetPort(savePort);                                 { and restore the state of thePort }

END; {of FUNC ClickedBtn}

{*****************************************************************************}
PROCEDURE DisplayMsg (theMsg: INTEGER; theType: TmsgType; doExit: BOOLEAN);
{display the dialog in my string list specified by (theMsg)}

VAR
   allDone     : BOOLEAN;
   dummy       : BOOLEAN;
   myEvent     : EventRecord;
   tempStr     : Str255;

BEGIN

{get the string specified by (theMsg) out of the string list}
```

```
GetIndString(tempStr, MyStrID, theMsg);

{remove the button from the dialog box}
HideControl(myBtnHdl);

SetIText(myTextHdl, tempStr);

IF (firstDialog) THEN
   BEGIN
   ShowWindow(myDlgPtr);
   DrawDialog(myDlgPtr);
   firstDialog := FALSE;
   END;

allDone := FALSE;

CASE theType OF
   WaitMsg:
      allDone := TRUE;
   StopMsg:
      BEGIN
      SetCTitle(myBtnHdl, btnOK^^);
      ShowControl(myBtnHdl);
      END;
   DiskMsg:
      BEGIN
      SetCTitle(myBtnHdl, btnCancel^^);
      ShowControl(myBtnHdl);
      END;
   END; {of CASE}

WHILE (NOT allDone) DO
   BEGIN
   SystemTask;
   dummy := GetOSEvent(everyEvent, myEvent);
   CASE myEvent.what OF
      DiskEvt:
         allDone := TRUE;
      MouseDown:
         IF ClickedBtn(myEvent) THEN
            IF (doExit) THEN
               ByeBye
            ELSE
               allDone := TRUE;
      END; {of CASE}
   END; {of WHILE}

END; {of PROC DisplayMsg}

{*****************************************************************************}
PROCEDURE DoMWInstall;
{ Do all the stuff to install the macworks image on the hard disk }

BEGIN

itemHit := NoteAlert(MWnoteAlert, NIL);   { tell user what's happening }
```

```
ForceOffLine;                       { make sony offline volume, eject it }
bootDiskOut := TRUE;                { jumping on thin ice ... }

GetMWDisk;                          { prompt for MW disk to copy (or bail out) }

HDControl(SetOffsetCode, Ptr(0), TRUE);   { set disk offset to 0 }                    {19Apr85}

IF (bigGuy) THEN                    { 1Meg Lisa, do it in one pass }
   BEGIN
   ReadDiskette(BothHalf);          { get 400K}
   WriteToDisk(BothHalf);           { we read 400K, so write it all out }
   END
ELSE                                { 512K Lisa, do it in two passes }
   BEGIN
   ReadDiskette(FirstHalf);         { get 200K }
   WriteToDisk(FirstHalf);          { write out the first 200K }
   ReadDiskette(LastHalf);          { suck up second half of sony }
   WriteToDisk(LastHalf);           { write out the second 200K }
   END;

WriteBootBlocks;                    { write out the bootblocks to the hard disk, set offset }

HideWindow(myDlgPtr);               { hide the window }
DisposDialog(myDlgPtr);             { get rid of generic dialog box }

ForceRemount;                       { get the Mac startup diskette back }

END; {of PROC DoMWInstall}

{*****************************************************************************}
PROCEDURE EjectDiskette;
{Hard eject a Sony diskette}

BEGIN
IF (PBEject(myMDPBptr, FALSE) <> 0) THEN
   Terminate(CantEject);

END; {of PROC EjectDiskette}

{*****************************************************************************}
PROCEDURE ForceOffline;
{Force sony off-line}

VAR
   err   : OSErr;

BEGIN

err := PBOffLine(myMDPBptr, FALSE);
EjectDiskette;

END; {of PROC ForceOffline}

{*****************************************************************************}
PROCEDURE ForceRemount;
```

```
{Force the user to insert the boot sony}

VAR
   err      : OSErr;
   fakeHdl  : Handle;

BEGIN
{asking for the resource from an off-line disk forces it on-line}

EjectDiskette;                          { get rid of MacWorks diskette }
fakeHdl := GetResource('RICH', 0);
bootDiskOut := FALSE;                   { Boot sony back in }

END; {of PROC ForceRemount}

{*****************************************************************************}
PROCEDURE GetMWDisk;
{ask the user for the original (source) MacWorks diskette}

VAR
   goodDisk : BOOLEAN;

BEGIN

goodDisk := FALSE;
REPEAT
   DisplayMsg(GetOldMW, DiskMsg, TRUE);
   IF (NOT MWDisk) THEN
      BEGIN {not a macworks diskette, so eject and let user bail out}
      EjectDiskette;
      DisplayMsg(NotMWDisk, StopMsg, FALSE);
      END
   ELSE
      goodDisk := TRUE;
   UNTIL goodDisk;

END; {of PROC GetMWDisk}

{*****************************************************************************}
PROCEDURE HDControl (theCode: INTEGER; theValue: Ptr; abort: BOOLEAN);
{do control calls to the hard disk}

TYPE
   TfakePBrec  = RECORD
                  temp1          : Ptr;
                  temp2          : INTEGER;
                  temp3          : INTEGER;
                  temp4          : Ptr;
                  ioCompletion   : ProcPtr;
                  temp6          : INTEGER;
                  ioNamePtr      : Ptr;
                  ioVRefNum      : INTEGER;
                  ioRefNum       : INTEGER;
                  csCode         : INTEGER;
                  csParam        : Ptr;
                 END; {of RECORD}
```

```
VAR
   err         : INTEGER;
   myCCPBlk    : TfakePBrec;
   myCCPBptr   : ParmBlkPtr;

BEGIN

myCCPBptr := ParmBlkPtr(@myCCPBlk);

WITH myCCPBlk DO
   BEGIN
   ioCompletion := nil;
   ioNamePtr := nil;
   ioVRefNum := HDVRefnum;
   ioRefNum := HDdrvrRefnum;
   csCode := theCode;
   csParam := theValue;
   END;

err := PBControl(myCCPBptr, FALSE);
IF (err <> 0) THEN
   IF (abort) THEN
      BEGIN
      CvtNum(err, resStrng);
      paramtext(resStrng,'','','');            { set error id for dialog display}
      Terminate(CantControl);
      END;

END; {of PROC HDControl}

{*****************************************************************************}
PROCEDURE HDStatus (VAR theStatus: parmBlkPtr);
{do status call to hard disk driver}

VAR
   err   : INTEGER;

BEGIN

WITH theStatus^ DO
   BEGIN
   ioCompletion := nil;
   ioRefnum := HDdrvrRefnum;     { hard disk driver refnum }
   ioVRefnum := HDVrefnum;       { hard disk default volume number }
   csCode := StatusCode;
   END;

err := PBStatus(theStatus, FALSE);
IF (err <> 0) THEN
   BEGIN
   CvtNum(err, resStrng);
   ParamText(resStrng,'','','');
   Terminate(CantControl);
   END;
```

```
END; {of PROC HDStatus}

{*****************************************************************************}
PROCEDURE InitBuffer;
{allocate 200K/400K buffer, set bigGuy flag if 1Meg Lisa}

VAR
   howMuch  : Size;

BEGIN
bigGuy := FALSE;
howMuch := 409600;                  {try for 400K}
copyBuffer := NewPtr(howMuch);
IF (copyBuffer <> NIL) THEN
   BEGIN
   bigGuy := TRUE;
   EXIT(InitBuffer);
   END;
howMuch := 204800;                  {try for 200K}
copyBuffer := NewPtr(howMuch);
IF (copyBuffer = NIL) THEN
   Terminate(CantInit);

END; {of PROC InitBuffer}

{*****************************************************************************}
PROCEDURE InitMyDialogs;

VAR
   i        : INTEGER;
   tempRect : Rect;
   myType   : INTEGER;

BEGIN

myDlgPtr := GetNewDialog(MyDlgID, NIL, Pointer(-1));
IF (myDlgPtr = NIL) THEN
   Terminate(CantInit);

GetDItem(myDlgPtr, 1, myType, Handle(myBtnHdl), tempRect);

IF (myBtnHdl = NIL) THEN
   Terminate(CantInit);

GetDItem(myDlgPtr, 2, myType, myTextHdl, tempRect);

IF (myTextHdl = NIL) THEN
   Terminate(CantInit);

btnOK := GetString(OKbtnID);
btnCancel := GetString(CancelbtnID);

firstDialog := TRUE;                    { for DisplayMsg stuff }

CouldDialog(MyDlgID);           { guarantee the dialog is in memory, since we eject the diskette }
```

```
END; {OF PROC InitMyDialogs}

{*****************************************************************************}
PROCEDURE MainInit;
{set up various mac managers}

BEGIN
InitGraf(@thePort);
InitCursor;
InitFonts;
InitWindows;
InitMenus;
TEInit;
InitDialogs(NIL);

watch := GetCursor(WatchID)^^;   {get the busy cursor}

FlushEvents(everyEvent, 0);      {clear out event queue}
myDlgPtr := NIL;                 {in case we bomb out in initialization}
myBtnHdl := NIL;

bootDiskOut := FALSE;

{set up hard disk & sony disk parameter blocks}

myHDPBptr := @myHDPBlk;
WITH myHDPBlk DO
   BEGIN
   ioNamePtr := NIL;
   ioRefNum  := HDdrvrRefnum;       { hard disk driver refnum }
   ioVRefNum := HDVRefnum;          { default volume for hard disk }
   ioPosMode := 1;                  { absolute positioning }
   ioPermssn := 0;                  { read/write permission }
   END;

myMDPBptr := @myMDPBlk;
WITH myMDPBlk DO
   BEGIN
   ioNamePtr := NIL;
   ioRefNum  := SonyDrvrRefnum;     { Sony driver refnum }
   ioVRefNum := SonyVRefnum;        { default drive number for internal sony }
   ioPosMode := 1;                  { absolute positioning }
   ioPermssn := 0;                  { read/write permission }
   END;

END; {of PROC MainInit}

{*****************************************************************************}
FUNCTION  MWDisk : BOOLEAN;
{returns TRUE if the first two blocks on the diskette look like Monitor format}

VAR
   firstWord   : INTEGER;
   tempPtr     : ^INTEGER;

BEGIN
```

```
ReadDiskette(TwoBlks);

{now check out the 1K in CopyBuffer to decide if it's monitor format}

tempPtr := Pointer(ORD(copyBuffer));
firstWord := tempPtr^;                          { get the first word }

IF (firstWord = $46FC) THEN                     { Monitor disk first word }
   MWDisk := TRUE
ELSE
   MWDisk := FALSE;

END; {of FUNC MWDisk}

{*****************************************************************************}
PROCEDURE ReadDiskette (readMode: TdiskMode);
{read from the sony, according to the mode.  If mode = TwoBlks & there was an
 error during the read, assume that the user inserted an uninitialized diskette,
 and stuff 0 in the first byte of {copyBuffer}, which will force a re-formatting
 of the diskette}

VAR
   blkCnt   : LongInt;
   blkLoc   : LongInt;

BEGIN
blkLoc := 0;
blkCnt := 400;
CASE readMode OF
   FirstHalf:
      BEGIN END;
   LastHalf:
      blkLoc := 400;
   BothHalf:
      blkCnt := 800;
   TwoBlks:
      blkCnt := 2;
   END; {of CASE}

WITH myMDPBlk DO
   BEGIN
   ioBuffer := copyBuffer;
   ioVRefnum := SonyVRefnum;        { reset it for safety }
   ioReqCount := blkCnt * 512;      { how much data to read }
   ioPosOffset := blkLoc * 512;     { where to read it from }
   END;

IF (readMode <> TwoBlks) THEN
   DisplayMsg(DoingRead, WaitMsg, FALSE);     {tell user we're reading the diskette}

SetCursor(watch);

IF (PBRead(myMDPBptr, FALSE) <> 0) THEN
   IF (readMode <> TwoBlks) THEN
      Terminate(CantRead)
   ELSE
```

```
      copyBuffer^ := 0;            {set first word to zero, force format for MW diskette}

SetCursor(arrow);

END; {of PROC ReadDisKette}

{*****************************************************************************}
PROCEDURE Terminate (why: INTEGER);
{tell user why, exit}

BEGIN

SetCursor(arrow);                { reset the cursor }

IF (myDlgPtr <> NIL) AND (myBtnHdl <> NIL) THEN
   DisplayMsg(why, StopMsg, TRUE)
ELSE
   ByeBye;

END; {of PROC Terminate}

{*****************************************************************************}
PROCEDURE WriteBootBlocks;
{write out the 8 blocks of boot code/volume map info}

TYPE
   TByte       = -128..127;                          { one byte of storage }

   TIntPtr     = ^INTEGER;

   TDiskType   = (ProfileDisk, WidgetDisk);

   TStatusInfo = RECORD
                  ioStuff  : ARRAY [0..27] OF TByte;     { all the ioCompletion, etc ParamBlock stuff }
                  error    : INTEGER;
                  info1    : INTEGER;
                  info2    : INTEGER;
                  DQEL     : ARRAY [0..13] OF Tbyte;
                  realSize : INTEGER;
                  HWDDRB   : TByte;
                  diskType : TByte;
                  restofit : ARRAY [0..20] OF TByte;
                 END; {of RECORD}

VAR
   codePtr     : TIntPtr;                    { pointer to block of code }
   fakePtr     : LongInt;
   i           : INTEGER;
   hDiskInfo   : TStatusInfo;
   tagBuf      : ARRAY [0..265] OF INTEGER;     { 532 bytes of header + data = 266 words }
   tagBufPtr   : Ptr;
   theDiskType : TDiskType;

BEGIN

{find out what type of disk is attached}
```

```
statusPBptr := @hDiskInfo;
HDStatus(statusPBptr);                    { find out what disk we're talking to }

IF (hDiskInfo.disktype <> 0) THEN         { must be a widget }
   theDiskType := WidgetDisk
ELSE
   theDiskType := ProfileDisk;

{Now comes some weird stuff.  At boot time, the Lisa boot Rom reads in the first 532
 bytes on the disk (block 0).  If the first 12 bytes are not all equal to $A (decimal 10),
 it signals a "Bad Bootblocks" error, otherwise it jumps to the 21st byte of the loaded
 block.  The block is loaded in with the 21st byte at location $20000.
 This works fine with a Profile, as when we write out the first block we make
 a control call to the hard disk driver telling it to use our tag buffer ptr (rather
 than the default, which points to Mac low memory).  We fill our tag buffer with
 all $A's, and off we go.  The widget, on the other hand, writes the header out at the
 END of each 512 byte block.  The boot Rom still expects the first 20 bytes to be the
 header, however, so we play the trick of having the first block of our boot code be
 20 $A's, then setting the tag buffer pointer to point to the 1st 20 bytes of the next
 block boot code.  Thus we create a Profile-style block 0 for loading, then reset all
 the pointers so that the rest of the blocks get written out correctly for a Widget/Profile.
 In actuality, only the first 12 bytes of the 20 byte header get written out correctly; next
 follows 7 bytes of $00 and 1 byte of checksum.  The Widget block 0 code, however, is far short
 of filling the block and is padded by an ascii string to be exactly 512, thus not writing
 out the last eight bytes isn't really a problem}

{create a tag buffer of 20 $AA's as the header of the first block}

FOR i := 0 TO 9 DO
   tagBuf[i] := -21846;                   { set to hex $AAAA }


IF (theDiskType = ProfileDisk) THEN
   BEGIN
   tagBufPtr := @tagBuf[0];               { normal tag buffer ptr }
   bootBlkPtr := @ProfileBoot;            { normal ptr to start of boot code }
   END
ELSE
   BEGIN
   bootBlkPtr := @tagBuf[0];              { first 20 bytes here are $A's }
   codePtr := TIntPtr(@WidgetBoot);       { point it to first byte of block 0 code }
   FOR i := 10 TO 265 DO                  { copy 512 bytes (256 words) of code }
      BEGIN
      tagbuf[i] := codePtr^;              { stuff a word into the buffer }
      codePtr := TIntPtr(ORD(codePtr) + 2);  { skip to next word }
      END;
   tagBufPtr := Ptr(ORD(@WidgetBoot) + 512); { just to point it somewhere }
   END;

HDControl(SetTagCode, tagBufPtr, TRUE);   { make the hard disk driver use my tag buffer }

WITH myHDPBlk DO
   BEGIN
   ioBuffer := bootBlkPtr;
   ioReqCount := 512;                     { write boot block 0 }
   ioPosOffset := 0;                      { write at beginning }
```

```
   END;

{now write out the first boot block}
IF (PBWrite(myHDPBptr, FALSE) <> 0) THEN
   Terminate(CantWrite);

{reset the tag buffer ptr to nil}
HDControl(SetTagCode, nil, TRUE);

{set up for writing out rest of boot blocks}
IF (theDiskType = ProfileDisk) THEN
   bootBlkPtr := Ptr(ORD(bootBlkPtr) + 512)  { set ptr to block 1 }
ELSE
   bootBlkPtr := tagBufPtr;                  { set ptr to block 1 }

WITH myHDPBlk DO
   BEGIN
   ioBuffer := bootBlkPtr;
   ioReqCount := (numBootBlks - 1) * 512;       { write rest of boot blocks }
   ioPosOffset := 512;                          { write at block 1 }
   END;

{now write out the rest of the boot blocks}
IF (PBWrite(myHDPBptr, FALSE) <> 0) THEN
   Terminate(CantWrite);

{Done writing to disk, so set up the correct offset}
fakePtr := LastMonBlk * 65536;                  { make longint w/high word = value of offset }
HDControl(SetOffsetCode, Ptr(fakePtr), TRUE);   { set offset of hard disk }

END; {of PROC WriteBootBlocks}

{*****************************************************************************}
PROCEDURE WriteToDisk (writeMode: TdiskMode);
{Write out the data from copyBuffer to the hard disk.  At this time, we haven't
 yet written out the bootblocks, so the offset is still 0 (absolute positioning)}

VAR
   blkCnt   : LongInt;          {number of 512 byte blocks to write}
   blkLoc   : LongInt;          {which block # to start writing to}

BEGIN
blkLoc := NumBootBlks;                  { skip boot blks, volume map blk }
blkCnt := 400;

CASE writeMode OF
   FirstHalf:
      BEGIN END;
   LastHalf:
      blkLoc := 400 + NumBootBlks;
   BothHalf:
      blkCnt := 800;
   END; {of CASE}

WITH myHDPBlk DO
   BEGIN
```

```
   ioBuffer := copyBuffer;              { transfer from the data buffer }
   ioReqCount := blkCnt * 512;          { how much data to write }
   ioPosOffset := blkLoc * 512;         { where to write to }
   END;

DisplayMsg(DoingWrite, WaitMsg, FALSE);         { tell user we're writing }

SetCursor(watch);

IF (PBWrite(myHDPBptr, FALSE) <> 0) THEN
   Terminate(CantWrite);

SetCursor(arrow);

END; {of PROC WriteToDisk}

{ **** Start of Hard Disk Install Procedures **** }

{*****************************************************************************}
Procedure ByeBye;

BEGIN

SetCursor(watch);                               { Set busy indicator }

IF (myDlgPtr <> nil) THEN
   BEGIN
   HideWindow(myDlgPtr);
   DisposDialog(myDlgPtr)
   END;

IF (bootDiskOut) THEN
   BEGIN
   EjectDiskette;                              { in case any MacWorks diskette is in the drive }
   ForceRemount;
   END;

GOTO 99;                                        { Jump to end of program }
END;

{*****************************************************************************}
Procedure CvtNum (num: INTEGER; VAR str: str255);

BEGIN
  if num < 0 then begin
    cvtnum(-num,str);
    str := concat('-',str);
    end
  else if num >= 10 then begin
    cvtnum(num div 10,str);
    str := concat(str,' ');
    str[length(str)] := chr((num mod 10) + ord('0'));
    end
  else begin
    str := ' ';
    str[1] := chr(num + ord('0'));
```

```
    end;
END;

{*****************************************************************************}
Procedure CheckErr (error: INTEGER);
{for errors we know about, display an alert & bail out, else assume benign and ignore}

BEGIN

CASE error OF

   NtLisaErr:
      itemhit := StopAlert(NtLisaAlert, NIL);   { Signal that this app can only be run on Lisa }

   NtRevCErr:
      itemHit := StopAlert(NtRevCAlert, nil);   {Signal that this app only runs under Rev C or later MacWorks}

   OnHDiskErr:
      itemHit := StopAlert(OnHDiskAlert, nil);  { Signal that this app can't run on the hard disk }

   OpenErr, ResrcErr, TimeErr, RespErr, PrtyErr, HardErr:  { Open, resource or control call failed }
      BEGIN
      cvtnum(error,resstrng);                   { convert error to string }
      paramtext(resstrng,'','','');             { set error id for alert display}
      itemhit := StopAlert(InitAlert, NIL)      { Signal error for debug purposes }
      END;

   NoDrvErr:                                    { No disk found }
      itemhit := StopAlert(NoDskAlert, nil);     { Display error }

   NoSpace:                                     { No blocks available }
      itemhit := StopAlert(SpcAlert, nil);       { Display error }

   OTHERWISE
      EXIT(CheckErr);

   END; {of CASE}

ByeBye;

END; {of PROC CheckErr}

{*****************************************************************************}
PROCEDURE DoInstall (dilogID: INTEGER; VAR error: INTEGER; drivetype: INTEGER);
{ Assumes dialog presented has two buttons, first is Cancel type, second is OK type }

VAR
   theDilog    : DialogPtr;

begin
  theDilog := GetNewDialog(dilogID, NIL, POINTER(-1));
  ModalDialog(NIL,itemhit);
  DisposDialog(theDilog);
  IF (itemhit = 2) THEN
      BEGIN
      error := HDFormat(driveType);              { volume comes back unmounted }
```

```
      IF (error <> 0) THEN
         EXIT(DoInstall);                          { return error code }
      IF (driveType <> LisaDisk) THEN
         DoMWInstall;                              { if not shared, blast MW image on }
      SetCursor(watch);
      error := HDName;                             { name the hard disk & write out the directory }
      SetCursor(arrow);
      CheckErr(error)                              { check for fatal error }
      END
   ELSE
      error := UserCancel;                         { user hit Cancel button }

END;  { of PROC DoInstall }

{*****************************************************************************}
PROCEDURE IsIconInSysRes;
{ check if icon already in system resource file; if not, install }

VAR
   theName3    : Str255;
   theName4    : Str255;
   theType3    : ResType;
   theType4    : ResType;
   myAttr3     : INTEGER;
   myAttr4     : INTEGER;
   theID3      : INTEGER;
   theID4      : INTEGER;
   myHandl3    : Handle;
   myHandl4    : Handle;

BEGIN
UseResFile(0);                          { switch to system resource file }
myhandl3 := GetResource('ICN#',ProIcon); { check if icon installed }
IF (myhandl3 = NIL) THEN
   BEGIN

   { First get Profile icon }
   UseResFile(AppRefID);                              { restore ptr to app's resource file }
   SetCursor(watch);                                  { set busy cursor }
   myhandl3 := GetResource('ICN#',ProIcon);           { get the ICN# handle }
   GetResInfo(myhandl3,theID3,thetype3,thename3);
   myattr3 := GetResAttrs(myhandl3);
   DetachResource(myhandl3);

   { Do same for internal disk icon }
   myhandl4 := GetResource('ICN#',IntrnlIcon);   { get the ICN# handle }
   GetResInfo(myhandl4,theID4,thetype4,thename4);
   myattr4 := GetResAttrs(myhandl4);
   DetachResource(myhandl4);

   UseResFile(0);                                 { switch to system resource file }

   AddResource(myhandl3,'ICN#',theID3,thename3); { add Profile icon }
   myattr3 := myattr3 + resChanged;               { set resChanged bit }
   SetResAttrs(myhandl3,myattr3);                 { write new attribute flags }
```

```
    AddResource(myhandl4,'ICN#',theID4,thename4); { add Internal disk icon }
    myattr4 := myattr4 + resChanged;              { set resChanged bit }
    SetResAttrs(myhandl4,myattr4);                { write new attribute flags }

    UpdateResFile(0);                             { write out the resources to system resource file}
    ReleaseResource(myhandl3);                    { release resources }
    ReleaseResource(myhandl4)
    END
ELSE
    ReleaseResource(myhandl3);                    { just release resource }

UseResFile(AppRefID);                             { restore cur res map as mine !!!! }

END; {of PROC IsIconInSysRes}

{*****************************************************************************}
PROCEDURE ReadBootBlks;
{read the Monitor bootblocks I (hopefully) just wrote to the hard disk}

VAR
    err   : INTEGER;

BEGIN

WITH myHDPBlk DO
    BEGIN
    ioBuffer := copyBuffer;
    ioRefNum  := HDdrvrRefnum;       { hard disk driver refnum }
    ioVRefNum := HDVRefnum;          { default volume for hard disk }
    ioReqCount := 512;               { read one block }
    ioPosOffset := 0;                { from the beginning }
    END;

err := PBRead(myHDPBptr, FALSE);

END; {of PROC ReadBootBlks}

{*****************************************************************************}
{start of main body of HDI (Hard Disk Install)}

BEGIN
MainInit;                                                                    Pg. 12
InitBuffer;                          { set up the copy buffer }              Pg  "
InitMyDialogs;                       { set up all the dialog pointers }      Pg  "

SetCursor(watch);
AppRefID := CurResFile;                 { save ptr to app's resource file }
result := HDOpen;                       { Open the driver, add drives to drive queue, try mount }  OPEN

{check for fatal errors}
SetCursor(arrow);
CheckErr(result);                                                            Pg 18

IF (result = MWorksFnd) THEN
    BEGIN                                  { if direct-start disk, ask how to proceed }    {19Apr85}
    LDlog := GetNewDialog(MWDlog, NIL, POINTER(-1));                                       {19Apr85}
```

```
   ModalDialog(NIL, itemhit);                                                    {19Apr85}
   DisposDialog(LDilog);                  { remove dialog }                      {19Apr85}

   CASE itemhit OF                                                               {19Apr85}
      1:                                  { Initialize button pressed }          {19Apr85}
        BEGIN                                                                    {19Apr85}
        DoInstall(ErasDilog, result, BlankDisk);   { erase entire drive }        {19Apr85}
        IF (result = UserCancel) THEN                                            {19Apr85}
           ByeBye;                                                               {19Apr85}
        END;                                                                     {19Apr85}

      2:                                  { Replace MacWorks button pressed }    {19Apr85}
        BEGIN                                                                    {19Apr85}
         DoMWInstall;                                                            {19Apr85}
        END;                                                                     {19Apr85}

      3:                                  { Cancel button pressed }              {19Apr85}
        ByeBye;                                                                  {19Apr85}
      END; {of CASE}                                                             {19Apr85}
   END {of IF THEN}                                                              {19Apr85}

ELSE IF (result = LisaFnd) THEN
   BEGIN                                  { Lisa OS format, ask about sharing }
   LDilog := GetNewDialog(LisaDilog, NIL, POINTER(-1));
   ModalDialog(NIL, itemhit);
   DisposDialog(LDilog);                  { remove dialog }

   CASE itemhit OF
      1:                                  { Share button pressed }
        BEGIN
        result := HDMount;                { Try to do mount of drive }
        CheckErr(result);                 { Check for fatal errors }
        IF (result = AllOK) THEN          { Mac side OK, say "already a shared disk"... }
           DoInstall(SharDilog, result, Lisadisk)
        ELSE IF (result < 0) THEN         { Mac side bad, say "unreadable, do you want to format".. }
           BEGIN
           DoInstall(BadSDilog,result,Lisadisk);
           IF (result = UserCancel) THEN
              ByeBye
           END;
        END;

      2:                                 { Erase All button pressed }
        BEGIN
        DoInstall(ErasDilog, result, BlankDisk);   { erase entire drive }
        IF (result = UserCancel) THEN
           ByeBye;
        END;

      3:                                 { Cancel button pressed }
        ByeBye;
      END; {of CASE}
   END {of IF THEN}

ELSE IF (result = AllOK) THEN             { Mac disk on-line }
   DOInstall(MacDilog, result, MacDisk)   { check if reinitialize wanted }
```

```
ELSE                                    { damaged or not initialized }
   BEGIN
   DoInstall(BadMDilog, result, BlankDisk);    { erase entire drive }
   IF (result = UserCancel) THEN
      ByeBye;
   END;

CASE (result) OF

   AllOK:
      BEGIN
      result := WrtBootBlks;                   { write boot blocks to hard disk }
      IsIconInSysRes;                          { make sure hard disk icons are in sysres }
      SetCursor(arrow);
      itemhit := NoteAlert(LstAlert, NIL);     { notify about moving system folder }
      END;

   MWorksFnd:                                  { MacWorks has been replaced - all done }    {19Apr85}
      BEGIN                                                                                   {19Apr85}
      END;                                                                                    {19Apr85}

   UserCancel:
      IsIconInSysRes;                          { make sure disk icons are in sysres }

   OTHERWISE
      BEGIN
      Cvtnum(result, resstrng);                { convert error to string }
      paramtext(resstrng, '', '', '');         { set error id for alert display}
      itemhit := StopAlert(InitAlert, NIL)     { Display error }
      END;
   END; {of CASE}

99:                                     { bail out point}
HideCursor;                             { return to the Finder . . .}

END.
```

```
;---------------------------------------------------------------------------------------
;  HARD DISK driver for MacWorks
;  (c) Apple Computer, Inc. 1984
;
;  Originator: Wendell Henry - February 10, 1984
;  Modification History:
;
;   2/23/84  RDC   Add driver header, change equates, add interleave tables
;   2/24/84  RDC   Generalize DiskOpen routine for drives on two port cards
;   2/29/84  RDC   Add DiskSync call when builtin parallel port in use
;   3/1/84   RDC   Add basic file tag support for read and writes
;   3/12/84  RDC   Remove for now search for drives other than builtin drive
;   3/14/84  RDC   Change DiskSync call to TRAPTO call
;   3/21/84  RDC   Add checks for driver already opened and drive installed in drive queue
;   3/29/84  RDC   Add support for icon control call, change eject call support
;   4/2/84   RDC   Add support for new fields in drive queue element (e.g., nonejectable disk)
;   4/3/84   RDC   Add driver support for Widget multi-block commands and interrupts
;   5/1/84   RDC   Change get icon call response for new Finder protocol
;   5/3/84   RDC   Add support for Lisa/Mac disk sharing
;   5/10/84  RDC   Add retry in initialization routine
;   5/15/84  RDC   Add erase cmd in DiskControl routine
;   5/16/84  RDC   Move drive queue check from DiskControl to Init controller routine
;   5/17/84  RDC   Change icon id
;   5/25/84  RDC   Add internal hard disk icon
;   6/11/84  RDC   Add intercept Eject command kluge
;   6/12/84  RDC   Add check for icon kluge
;   6/13/84  RDC   Add erase cmd kluge
;   6/13/84  RDC   Add support for DiskStatus call
;   6/14/84  RDC   Do cleanup of driver portion
;   6/18/84  RDC   Modify erase cmd kluge
;   6/21/84  RDC   Change driver name to .HardDisk
;   7/5/84   RDC   Fix bug in error recovery routine S40
;
;  Notes:  add control call support to check for other disks
;---------------------------------------------------------------------------------------

            .NOLIST
            .INCLUDE TLASM-SYSEQU.TEXT
            .INCLUDE TLASM-FSEQU.TEXT
            .INCLUDE TLASM-SYSERR.TEXT
            .INCLUDE TLASM-SYSMACS.TEXT
            .INCLUDE TLASM-TOOLMACS.TEXT
            .LIST

;
;Timeout values
STRTTIME        .EQU      $1200000            ;Powerup time = about 180 sec
RSPTIME         .EQU      $180000             ;Normal response time = about 16 secs
RDTIME          .EQU      $000C0000           ;READ TIMEOUT.  ABOUT 8 SECONDS
;RSPTIME         .EQU      $0050               ;RESPONSE TIMEOUT = ABOUT 0.5 MS
COUNTLIMIT      .EQU      100                 ;TIMEOUT LIMIT

;Error codes
READ_ERR        .EQU      ReadErr             ;Error during disk read
WRITE_ERR       .EQU      WritErr             ;Error during disk write
TIME_ERR        .EQU      -95                 ;Response timeout error
```

```
NODISK_ERR      .EQU    NoDriveErr          ;No disk attached
RESP_ERR        .EQU    -96                 ;Response error
CS_ERR          .EQU    BadDCkSum           ;Checksum error
PRTY_ERR        .EQU    -97                 ;Parity error
HD_ERR          .EQU    -98                 ;Hard Error

WAIT_INT        .EQU    1                   ;Wait for interrupt indicator

;  Equates for parallel port 6522

VIABASE         .EQU    $00FCD901           ;Base address
IRB             .EQU    0
ORB             .EQU    0
IRA             .EQU    8
ORA             .EQU    8
DDRB            .EQU    $10
DDRA            .EQU    $18
T2CL            .EQU    $40
T2CH            .EQU    $48
ACR             .EQU    $58
PCR             .EQU    $60
IFR             .EQU    $68
IER             .EQU    $70
PORTA           .EQU    $78
RESETC          .EQU    $580                ;location to reset controller

;  Equates for Keyboard 6522
VIA2BASE        .EQU    $00FCDD81           ;Base address
ORB2            .EQU    $0                  ;Port B output register
DDRB2           .EQU    $4                  ;Port B Data direction register

;  Equates for Two port cards
TWOPORT         .EQU    $E002               ;Two port card id
SLOT1CH1        .EQU    $00FC2000           ;Slot 1, channel 1
SLOT1CH2        .EQU    $00FC2800           ;Slot 1, channel 2
SLOT2CH1        .EQU    $00FC6000           ;Slot 2, channel 1
SLOT2CH2        .EQU    $00FC6800           ;Slot 2, channel 2
SLOT3CH1        .EQU    $00FCA000           ;Slot 3, channel 1
SLOT3CH2        .EQU    $00FCA800           ;Slot 3, channel 2
DDRB3           .EQU    $10                 ;offset for DDRB on 2-port card VIA

;WIDGET SPARE TABLE OFFSETS
FMT_INTL        .EQU    9                   ;OFFSET TO DISK INTERLEAVE
INTL_MAP        .EQU    454                 ;OFFSET TO SOFT INTERLEAVE MAP

;WIDGET SOFT INTERLEAVE
W_IMAP          .EQU    $000C0511           ;FIRST PART OF INTERLEAVE TABLE FOR 5:1

;  Equates for low memory space used by boot ROM
MWBASE          .EQU    $500                ;offset in MacWorks world
SLOT1ID         .EQU    $298                ;card id in slot 1
SLOT2ID         .EQU    $29A                ;card id in slot 2
SLOT3ID         .EQU    $29C                ;card id in slot 3

HDSKVARS        .EQU    TwiggyVars          ;low mem ptr to disk driver locals
```

```
;  Drive specific local variables (offsets from driveX)
HWBASE          .EQU    0                       ;Hardware base address for data lines (=0 if not installed)
HWRESET         .EQU    HWBASE+4                ;Hardware base address for reset lines
INFO1           .EQU    HWRESET+4               ;Write protect/disk in place info
INFO2           .EQU    INFO1+2                 ;Install/sides info
DQEL            .EQU    INFO2+2                 ;Drive queue element (14 bytes)
REALSIZE        .EQU    DQEL+14                 ;actual size of entire disk (as opposed to just Mac portion)
HWDDRB          .EQU    REALSIZE+2              ;offset for VIA DDRB register
DRIVETYPE       .EQU    HWDDRB+1                ;Type of disk drive (<>0 is Widget)
ER_HS           .EQU    DRIVETYPE+1             ;Expected response during next handshake
COMMAND         .EQU    ER_HS+1                 ;Current driver service command
IOCOMMAND       .EQU    COMMAND+1               ;Requested I/O command (read/write)
SECTOR          .EQU    COMMAND+1               ;Current sector being accessed (high byte unused)
RETRYCNT        .EQU    SECTOR+4                ;Retry count for errors
SPARECNT        .EQU    RETRYCNT+1              ;Sparing threshold
CMD_BUFFER      .EQU    IOCOMMAND               ;Buffer for command bytes/sector/retrycnt/sparecnt

ERROR           .EQU    SPARECNT+1              ;Error for current operation
IOERROR         .EQU    ERROR+2                 ;Error return
STATE           .EQU    IOERROR+2               ;Current state of driver state machine
STATUS          .EQU    STATE+2                 ;4 bytes for reading disk status
FIRSTFSEQ       .EQU    STATUS+4                ;first file sequence # for current read/write request
FSBLKSDONE      .EQU    FIRSTFSEQ+2             ;# of blocks read/written for current request
SECT_LEFT       .EQU    FSBLKSDONE+2            ;# of blocks remaining for current request
CHECKSUM        .EQU    SECT_LEFT+4             ;save of precomputed checksum for writes (in high byte)
CXFERCNT        .EQU    CHECKSUM+2              ;current transfer count
CSUM_VALID      .EQU    CXFERCNT+1              ;checksum valid indicator
NESTED_BDR      .EQU    CSUM_VALID+1            ;indicator for bad response received
ACCSTAT         .EQU    NESTED_BDR+2            ;accumulated state register status
CMD_BUF         .EQU    ACCSTAT+4               ;command buffer for Widget commands
CNTRESETS       .EQU    CMD_BUF+8               ;number of times Widget reset
STRTBLK         .EQU    CNTRESETS+2             ;starting block for Mac disk area
DrvLclLth       .EQU    STRTBLK+2               ;length of drive local area

;  Driver local variables
DCE             .EQU    0                       ;Pointer to Device control entry
DRIVE           .EQU    DCE+4                   ;Drive #, 4 = default disk
EJCTSUB         .EQU    DRIVE+2                 ;ptr to normal eject routine
INITRQST        .EQU    EJCTSUB+4               ;initialize request
PAD             .EQU    INITRQST+1
DRIVE4          .EQU    PAD+1                   ;Locals for internal drive

;DRIVE5          .EQU     DRIVE4+DrvLclLth      ;Locals for drive on slot1, chan1
;DRIVE6          .EQU     DRIVE5+DrvLclLth      ;Locals for drive on slot1, chan2
;DRIVE7          .EQU     DRIVE6+DrvLclLth      ;Locals for drive on slot2, chan1
;DRIVE8          .EQU     DRIVE7+DrvLclLth      ;Locals for drive on slot2, chan2
;DRIVE9          .EQU     DRIVE8+DrvLclLth      ;Locals for drive on slot3, chan1
;DRIVE10         .EQU     DRIVE9+DrvLclLth      ;Locals for drive on slot3, chan2

TagBufPtr       .EQU    DRIVE4+DrvLclLth        ;ptr to 12 bytes for disk header if nonzero
DiskVarLth      .EQU    TagBufPtr+4             ;size of local variable space

; Drive numbers for attached hard disks
DRV4            .EQU    4                       ;drive on builtin port
DRV5            .EQU    5                       ;drive on slot 1, lower port
DRV6            .EQU    6                       ;drive on slot 1, upper port
```

```
DRV7            .EQU    7               ;drive on slot 2, lower port
DRV8            .EQU    8               ;drive on slot 2, upper port
DRV9            .EQU    9               ;drive on slot 3, lower port
DRV10           .EQU    10              ;drive on slot 3, upper port

; Offsets into block -1 for identifying disk
DEVTYPE         .EQU    14              ;Type of disk
SZ              .EQU    18              ;Start of 3 byte disk size

; Offsets into Lisa OS page 0 block
LASTBLK         .EQU    $70             ;Last block used
BLKSZ           .EQU    $7C             ;Block size/data size in bytes
SZPARM          .EQU    $02180200       ;value for block size/data size for Profile/Widget

; Driver service commands
INITCMD         .EQU    0               ;Initialize controller
IOCMD           .EQU    2               ;Perform I/O

; Drive commands
READCMD         .EQU    2               ;Read
WRITCMD         .EQU    3               ;Write

; Misc constants
OCD             .EQU    0               ;Open cable detect line
BSY             .EQU    1               ;Controller busy line
RETRIES         .EQU    4               ;I/O retry count
MAXRETRY        .EQU    3               ;Max controller retry count
SPARE           .EQU    10              ;Sparing threshold
MAXCNT          .EQU    255             ;maximum block count for Widget multi-block

;Control codes
VfyCode         .EQU    5               ;verify disk
FmtCode         .EQU    6               ;format disk
TagCode         .EQU    8               ;set tag buffer
IconCode        .EQU    20              ;get icon
InitDCode       .EQU    30              ;init device
EraseCode       .EQU    31              ;erase (format) disk

; Misc equates

_DiskSync       .EQU    108             ;trap # for disksync call
ProIcon         .EQU    -16351          ;ID for external drive icon
IntrnlIcon      .EQU    -16350          ;ID for internal drive icon
StopEject       .EQU    8               ;id for non-ejectable disk
EjctTrap        .EQU    $17             ;trap # for eject call
EraseTyp        .EQU    CSParam+4       ;erase type for format call


                .PROC   HDDRVR,0

                .DEF    DISKOPEN, DISKPRIME, DISKSTATUS, DISKCONTROL, DISKCLOSE


; driver header:

HdskDrvr
```

```
            .WORD   $4F00               ; Ram-based,read,write,Ctrl,Status
            .WORD   0,0                 ; no delay or EMask
            .WORD   0                   ; no menu

; Entry point offset table

            .WORD   DiskOpen-HdskDrvr    ; open
            .WORD   DiskPrime-HdskDrvr   ; prime
            .WORD   DiskControl-HdskDrvr ; control
            .WORD   DiskStatus-HdskDrvr  ; status
            .WORD   DiskClose-HdskDrvr   ; close
            .BYTE   9
            .ASCII  '.HardDisk'

;-------------------------------------------------------------------------------
;
; DiskOpen
;
; Arguments:  A0 (input)  -- pointer to caller's parameter block
;             A1 (input)  -- pointer to device control entry
;             D0 (output) -- result code
;
;-------------------------------------------------------------------------------

DiskOpen    TST.L   DctlStorage(A1)     ;locals already allocated?
            BNE.S   @2                  ;skip if yes

@1          MOVE.L  #DiskVarLth,D0      ;Get memory for driver locals
            _NewPtr ,SYS,CLEAR
            MOVE.L  A0,DctlStorage(A1)  ;Keep pointer in DCE
            MOVE.L  A0,HDSKVARS         ;and save in low mem for other use
            MOVE.L  A1,DCE(A0)          ;Keep pointer to device control entry

;  setup special eject code

            MOVE.L  A1,-(SP)            ;save DCE ptr
            MOVE.L  A0,A1               ;save locals ptr
            MOVE.W  #EjctTrap,D0        ;first get trap address
            _GetTrapAddress
            MOVE.L  A0,EjctSub(A1)      ;save it
            LEA     HDEject,A0          ;set ptr for our eject routine
            MOVE.W  #EjctTrap,D0        ;replacing eject trap
            _SetTrapAddress
            MOVE.L  (SP)+,A1            ;restore DCE ptr

@2          RTS

;-------------------------------------------------------------------------------
;  Substitute Eject routine for hard disk
;-------------------------------------------------------------------------------

HDEject     MOVEM.L D1/A2,-(SP)         ;save regs
            MOVE.W  IOVRefNum(A0),D0    ;get drive for eject call
            BSR.S   CkDrvParm

            MOVE.L  HDSKVARS,A1         ;get ptr to locals
```

```
            MOVE.L  DCE(A1),A2          ;get DCE ptr
            CMP.W   DCtlRefNum(A2),D0   ;request for a hard disk?
            BNE.S   @1
            MOVE.W  #ControlErr,D0      ;signal error
            MOVEM.L (SP)+,D1/A2         ;restore regs
            RTS                         ;and return to caller

@1          MOVE.L  EjctSub(A1),A1      ;get ptr to real eject routine
            MOVEM.L (SP)+,D1/A2         ;restore regs
            JMP     (A1)                ;and go do it

;---------------------------------------------------------------------------
;  Routine to check drive parameter, translate as appropriate
;  D0 = drive parm
;---------------------------------------------------------------------------

CkDrvParm   TST     D0                  ;check drive type parm
            BEQ.S   @1                  ;skip if no drive/refnum specified
            BMI.S   @2                  ;skip if really a VrefNum
            BSR.S   GetVCBDrv           ;else must be a drive #
            BRA.S   @3                  ;go do compare

@1          MOVE.L  DefVCBPtr,D1        ;try default VCB ptr
            BEQ.S   @3                  ;skip if none
            MOVE.L  D1,A1               ;get ptr
            MOVE.W  VCBDRefNum(A1),D0   ;get driver refnum
            BRA.S   @3

@2          BSR.S   GetVCBRef           ;go get driver refnum

@3          RTS                         ;go check it

;---------------------------------------------------------------------------
; Routine:      GetVCBRfn,GetVCBDrv
; Arguments:      D0.W (input)  -- IODrvNum(A0) = IOVRefNum(A0)
;                 D0.W (output) -- 0 (no such volume) or driver refnum
;                 All other regs are preserved
; Function:     Determine driver refnum from VCB with DriveNum or VRefNum field.
;
;---------------------------------------------------------------------------

GetVCBRef   MOVEM.L D1-D2/A0,-(SP)
            MOVEQ   #VCBVRefNum,D1      ; looking for VCB by volume refnum
            BRA.S   GetVCB1

GetVCBDrv   MOVEM.L D1-D2/A0,-(SP)
            MOVEQ   #VCBDrvNum,D1       ; looking for VCB by drive number

GetVCB1     LEA     VCBQHdr,A0          ; search the queue of VCBs
            MOVE.L  QHead(A0),D2        ; get first element

@1          BEQ.S   @3                  ; exit if no match
            MOVE.L  D2,A0               ; next element pointer
            CMP.W   0(A0,D1),D0         ; match?
            BEQ.S   @2                  ; then take the good exit . . .
            MOVE.L  QLink(A0),D2
```

```
            BRA.S   @1

@2          MOVE.W  VCBDRefNum(A0),D0   ;get driver refnum
@3          MOVEM.L (SP)+,D1-D2/A0
            RTS

;---------------------------------------------------------------------------
;
; DiskControl
;
; Arguments:  A0 (input)  -- pointer to caller's parameter block
;
;               CSCode = 1 for KillIO
;                        5 for Verify
;                        6 for Format
;                        7 for Eject
;                        8 for Tag Buffer set
;                       20 for Get Icon
;                       30 for Init controller and add to drive queue
;                       31 for Erase entire disk
;
;               CSParam = ptr to icon area for Get icon call
;
;             A1 (input)  -- pointer to device control entry
;             D0 (output) -- result code
;
; Uses:       A2 -- save of local storage pointer
;             A3 -- ptr to drive specific variables
;             A4 -- ptr to disk port base address
;             A5 -- ptr to disk reset/parity line address
;             A6 -- ptr to driver variables
;
;
; To add: support for KillIO, Set Tag Buffer calls
;---------------------------------------------------------------------------

DiskControl BSR     CkDrvNum            ;ensure proper drive #
            MOVE.L  DCtlStorage(A1),A2  ; get ptr to locals
            MOVE.W  IODrvNum(A0),Drive(A2)  ; save drive #
            MOVE.W  CSCode(A0),D1
            SUBQ.W  #KillCode,D1        ; KillIO?
            BNE.S   @1
            MOVE    SR,-(SP)            ; turn into NOP for now
            RTE                         ; special return for KillIO

;**VERIFY

@1          SUBQ.W  (#VfyCode-KillCode),D1   ; Verify?
            BNE.S   @2
            BRA     NoCmd                    ; No support for now

;**FORMAT

@2          SUBQ.W  (#FmtCode-VfyCode),D1    ; Format?
            BNE.S   @3
            BRA     @9                       ; go do simple erase



```

```
;**EJECT

@3          SUBQ.W  (#EjectCode-FmtCode),D1  ; Eject?
            BNE.S   @4
            BRA     NoCmd                    ; Can't eject a hard disk!

;**SET TAG BUFFER

@4          SUBQ.W  (#TagCode-EjectCode),D1  ; Tag Buffer set?
            BNE.S   @5
            BRA     NoCmd                    ; No support for now

;**GET ICON

@5          SUB.W   (#IconCode-TagCode),D1   ; Get icon?
            BNE.S   @7

;  first do check to see what drive we'er operating from

            MOVE.W  BootDrive,D0             ; get current drive
            BSR     CkDrvParm                ; go translate to driver refnum
            CMP.W   DCtlRefNum(A1),D0        ; is it the hard disk?
            BNE     NoCmd                    ; exit if it's not

            MOVEM.L A3/A6,-(SP)              ; save regs
            BSR     GETLCLS                  ; get drive locals ptr
            TST.B   DriveType(A3)            ; internal drive?
            BEQ.S   @13                      ; skip if not
            MOVE.W  #IntrnlIcon,D0           ; else use internal icon
            BRA.S   @14
@13         MOVE.W  #ProIcon,D0              ; use external icon
@14         MOVE.W  D0,CSParam(A0)           ; return icon id to Finder
            MOVEQ   #0,D0                    ; no errors
            MOVEM.L  (SP)+,A3/A6             ; restore regs
            BRA     Done                     ; and return

;**INIT DEVICE CONTROLLER

@7          SUB.W   (#InitDCode-IconCode),D1 ; Init device?
            BNE     @8

;   Check for disk requested

            MOVEM.L  A3-A6,-(SP)          ;save regs
            BSR      GETLCLS              ;get drive locals ptr

            MOVE.L   #VIABASE,A4          ;Hardware base address for data lines
            MOVE.L   #VIA2BASE,A5         ;Hardware base address for reset lines
            MOVE     D0,D1                ;set drive #
            BSR      CHK4DISK             ;go check drive type and enter in drive queue
                                          ; if device is disk
            TST      D0                   ;any errors?
            BEQ.S    @11                   ;skip if none
            MOVEM.L  (SP)+,A3-A6          ;else restore regs
            BRA      DONE                 ;and exit
```

```
;  disk found - now setup dummy request block, buffer area for check if Lisa disk

@11         MOVEM.L  A0/A2,-(SP)            ;save other regs
            SUBA     #IOQELSIZE,A7          ;Stack space for dummy I/O request
            MOVE.L   A7,A0

            MOVEQ    #127,D0                ;clear buffer space area for block data
@12         CLR.L    -(SP)
            DBF      D0,@12

            MOVE.L   A7,IOBUFFER(A0)        ;save buffer address
            MOVE.W   #READCMD,IOTRAP(A0)    ;do read
            MOVEQ    #38,D0                 ; of page 0 block for Lisa OS
            CLR      STRTBLK(A3)            ; with no offset
            MOVEQ    #2,D1                  ;set D1 = 512
            ASL.L    #8,D1
            MOVE.L   D1,IOBYTECOUNT(A0)     ;read single block
            MULU     D1,D0                  ;translate block # to byte offset
            MOVE.L   D0,IOPOSOFFSET(A0)     ;save as read position
            MOVE.W   #1,IOPOSMODE(A0)       ;from start of disk
            BSR      CALLDRIVER             ;Go do it, result in D0
            TST      D0                     ;check for error
            BNE.S    @20                    ;skip if yes

            MOVE.L   IOBUFFER(A0),A2        ;get buffer address
            CMP.L    #SZPARM,BLKSZ(A2)      ;check block size/data size parms
            BNE.S    @20                    ;skip if not correct
            MOVE.L   LASTBLK(A2),D0         ;get last block used value
            ADD.L    #9,D0                  ;add # of unused blocks
            MOVEQ    #DQEL,D1               ;length of drive queue element
;            CMP.W    DQDRVSIZE(A3,D1),D0    ;blocks used same as drive size?
;            BGE.S    @15                    ;skip if no blocks available
            MOVE     D0,STRTBLK(A3)         ;else save as first block for Mac area
            SUB.W    D0,DQDRVSIZE(A3,D1)    ;and set correct value for Mac disk total blocks
@15         MOVE     #LisaDskErr,D0         ;set Lisa disk found return code

@20         ADDA     #(IOQELSIZE+512),A7    ;Pop space for sector and I/O request
            MOVEM.L  (SP)+,A0/A2            ;restore regs
            MOVEM.L  (SP)+,A3-A6
            BRA.S    DONE                   ;and exit with result

;**ERASE DISK

@8          SUB.W    (#EraseCode-InitDCode),D1 ; Erase (format) disk?
            BNE.S    NoCmd
            MOVE.W   EraseTyp(A0),D0          ; erase entire disk?
            BNE.S    @9                       ; skip if not

            MOVEM.L  A3/A6,-(SP)              ; else save regs
            BSR.S    GETLCLS                  ; get ptr to locals
            MOVE     STRTBLK(A3),D0           ; get start block
            MOVEQ    #DQEL,D1                 ; get offset to drive queue element
            ADD.W    D0,DQDRVSIZE(A3,D1)      ; update disk size
            CLR      STRTBLK(A3)              ; and set starting block as block 0
            MOVEM.L  (SP)+,A3/A6              ; restore regs
```

```
;    setup dummy request block to do format

@9          MOVE.L   A0,-(SP)              ;save regs
            SUBA     #IOQELSIZE,A7         ;Stack space for dummy I/O request
            MOVE.L   A7,A0

            MOVEQ    #127,D0               ;clear buffer space area for block data
@10         CLR.L    -(SP)
            DBF      D0,@10

            MOVE.L   A7,IOBUFFER(A0)       ;save buffer address
            MOVE.B   #1,INITRQST(A2)       ;request zero operation
            MOVE.W   #WRITCMD,IOTRAP(A0)   ;via write command
            MOVEQ    #DQEL,D0              ;get drive size in blocks
            ADD.L    #DRIVE4,D0            ;ptr to drive locals (assume drive# = 4)
            MOVE.W   DQDrvSize(A2,D0),D0
            MOVEQ    #2,D1                 ;set D1 = 512
            ASL.L    #8,D1
            MULU     D1,D0                 ;translate block count to byte count
            MOVE.L   D0,IOBYTECOUNT(A0)    ;Write entire disk
            MOVEQ    #0,D0                 ;starting with first block
            MOVE.L   D0,IOPOSOFFSET(A0)
            MOVE.W   #1,IOPOSMODE(A0)      ;from start of disk
            BSR      CALLDRIVER            ;Go do it, result in D0
            CLR.B    INITRQST(A2)          ;remove init request
            ADDA     #(IOQELSIZE+512),A7   ;Pop space for sector and I/O request
            MOVE.L   (SP)+,A0              ;restore regs
            BRA.S    DONE                  ;and exit

; Invalid command

NoCmd       MOVE     #ControlErr,D0          ; unsupported command

Done        BRA      DiskDone


;-----------------------------------------------------------
;  Subroutine to compute ptrs to drive locals
;  Returns:  A3 - ptr to drive specific locals
;            A6 - ptr to driver locals
;  Destroys: D1,D2

GETLCLS
        MOVE.L  HdskVars,A3     ;get ptr to locals
        MOVE.L  A3,A6           ;save as global ptr
        ADD.L   #DRIVE4,A3      ;set ptr to start of drive locals
        CLR.L   D1              ;clear for use
        CLR.L   D2
        MOVE.W  Drive(A6),D1    ;get drive # currently active
        SUBQ    #Drv4,D1        ;convert to multiplier
        MOVE    #DrvLclLth,D2   ;get length of drive locals
        MULU    D2,D1           ;compute offset to drive's locals
        ADD.L   D1,A3           ;and set ptr to specific drive's locals
        RTS
```

```
;---------------------------------------------------------------------------
;
; CHK4DISK - check for disk attached to designated port
;
; Arguments:  A0 (input)  -- pointer to caller's parameter block
;             A1 (input)  -- pointer to device control entry
;             A3 (input)  -- pointer to drive specific locals
;             A4 (input)  -- pointer to port base address data lines
;             A5 (input)  -- pointer to port base address reset lines
;             A6 (input)  -- pointer to driver locals
;             D0 (output) -- result code
;             D1 (input)  -- drive #
;
;---------------------------------------------------------------------------

CHK4DISK    MOVE.L   A0,-(SP)             ;save reg

;  check to see if there really is a drive attached

@4          MOVE.L   A4,HWBASE(A3)        ;Hardware base address for data lines
            MOVE.L   A5,HWRESET(A3)       ;Hardware base address for reset lines
            MOVE.B   #DDRB2,HWDDRB(A3)    ;assume not using slots
            CMP.L    #VIABASE,A4          ;are we right?
            BEQ.S    @0                   ;skip if yes
            MOVE.B   #DDRB3,HWDDRB(A3)    ;set for slot access

@0          MOVE.W   D1,DRIVE(A6)         ;save drive # in use
            MOVE.B   #INITCMD,COMMAND(A3) ;Request initialization
            BSR      PRODRIVER            ;Go execute initialization function
            MOVE.W   ERROR(A3),D0         ;check for error
            BEQ.S    @99                  ;skip if all OK
            CMP      #NODISK_ERR,D0       ;no disk error?
            BNE.S    @99                  ;skip if not to exit
            CLR.W    ERROR(A3)            ;remove error
            CLR.L    HWBASE(A3)           ;set for no attached disk

@99         MOVE.L   (SP)+,A0             ;restore reg
            RTS

;---------------------------------------------------------------------------
;
; DiskStatus
;
; Arguments:  A0 (input)  -- pointer to caller's parameter block
;             A1 (input)  -- pointer to device control entry
;             D0 (output) -- result code
;
; To add: return of block -1 parms?
;---------------------------------------------------------------------------

DiskStatus      MOVEQ   #StatusErr,D0          ; set error
                CMP.W   #DrvStsCode,CSCode(A0) ; valid status call?
                BNE.S   DiskDone               ; skip if error
                BSR     CkDrvNum               ; else check if for correct drive

                MOVEM.L A1/A3/A6,-(SP)         ; save regs
```

```
                BSR     GETLCLS                 ; get ptr to locals
                MOVE.L  A3,A1                   ; get drive specific ptr
                ADD.L   #INFO1,A1               ; set ptr to start area
                LEA     CSParam(A0),A0          ; get ptr to parameter block area

                MOVE.W  ERROR(A3),(A0)+         ; return last error as first parm
                MOVEQ   #(DriveType-INFO1)/2,D0 ; set count for next set of parms
@1              MOVE.W  (A1)+,(A0)+             ; stuff them in param block
                SUBQ.W  #1,D0                   ; leave D0 = 0
                BNE.S   @1

                MOVE.L  CNTRESETS(A3),(A0)+     ; return reset cntr as last parm
                MOVEM.L (SP)+,A1/A3/A6          ; restore regs and exit


;---------------------------------------------------------------------
;
; Routine:      DiskDone
;
; Arguments:        D0 (input)  -- last error
;                   A1 (output) -- pointer to disk DCE
;
;---------------------------------------------------------------------

DiskDone
            MOVE.W  D0,DskErr           ; save last error for file system
            MOVE.L  JIODone,-(SP)
DiskRTS     RTS                         ; use IODone vector

;-------------------------------------------------------------------------
;
; DiskClose
;
; Arguments:  A0 (input)  -- pointer to caller's parameter block
;             A1 (input)  -- pointer to device control entry
;             D0 (output) -- result code
;
;-------------------------------------------------------------------------

DiskClose     RTS             ;Just return - no errors, not closable


;-------------------------------------------------------------------------
;
; DiskPrime
;
; Arguments:  A0 (input)  -- pointer to caller's parameter block
;             A1 (input)  -- pointer to device control entry
;             D0 (output) -- result code
;
;-------------------------------------------------------------------------

DiskPrime
              BSR       CKDrvNum            ;ensure proper drive #
              BSR.S     CallDriver
              BRA.S     DiskDone            ;do return via IODONE
```

```
;---------------------------------------------------------------------
;  subroutine to handle interface for PRODRIVER routine
;---------------------------------------------------------------------

CALLDRIVER
            MOVEM.L  D2-D4/A0-A6,-(SP)     ;Save registers
            MOVE.W   IOTRAP(A0),D4         ;I/O command
            AND.W    #$FF,D4               ;extract I/O command
            SUBQ.B   #2,D4                 ;0=read, 1=write, 2=write verify, 3=format
            MOVE.L   DctlStorage(A1),A3    ;Get local vars
            MOVE.L   A3,A6                 ;save ptr
            ADD.L    #DRIVE4,A3            ;set ptr to start of drive locals
            CLR.L    D1                    ;clear for use
            CLR.L    D2
            MOVE.W   Drive(A6),D1          ;get drive #
            SUBQ     #Drv4,D1              ;convert to multiplier
            MOVE     #DrvLclLth,D2         ;get length of drive locals
            MULU     D2,D1                 ;compute offset to drive's locals
            ADD.L    D1,A3                 ;and set ptr to specific drive's locals

            MOVE.L   IOByteCount(A0),D2    ;# bytes requested
            LSR.L    #7,D2                 ;Convert byte count to block count
            LSR.L    #2,D2
            MOVE.L   D2,SECT_LEFT(A3)      ;save as beginning block count
            MOVE.L   DCtlPosition(A1),D3   ;get current byte position
            MOVE.W   IOPosMode(A0),D0      ;get positioning mode
            ROXR.B   #2,D0                 ;place bits into sign, carry
            BPL.S    @3                    ;skip if no offset (0 or 2)
            BCC.S    @2                    ;mode 1 means absolute
            ADD.L    IOPosOffset(A0),D3    ;mode 3 is relative offset
            BRA.S    @3
@2          MOVE.L   IOPosOffset(A0),D3    ;use as starting byte position
@3          ASR.L    #7,D3                 ;Convert byte position to blk position
            ASR.L    #2,D3
            ADD      STRTBLK(A3),D3        ;add starting block offset
            MOVE.L   D3,SECTOR(A3)         ;save as starting sector

            MOVE.B   D4,IOCOMMAND(A3)      ;save command
            MOVE.L   IOBuffer(A0),A2       ;buffer pointer for data
            MOVE     #TagData+2,A1         ;assume use of default tag buffer area
            TST.L    TAGBUFPTR(A6)         ;separate tag buffer exists?
            BEQ.S    @1                    ;skip if not
            MOVE.L   TAGBUFPTR(A6),A1      ;else use it
@1          MOVE.W   TAGDATA+8,FIRSTFSEQ(A3) ;save first file seq # for writes
            CLR.W    FSBLKSDONE(A3)        ;zero blocks done field

            MOVE.B   #MAXRETRY,RETRYCNT(A3) ;Initialize retry/spare count fields
            MOVE.B   #SPARE,SPARECNT(A3)
            MOVE.B   #IOCMD,COMMAND(A3)    ;request I/O service
            BSR.S    PRODRIVER             ;Start I/O operation
            MOVE.W   ERROR(A3),D0          ;set for exit
            MOVE.W   D0,IOERROR(A3)        ;Save as last error code
            MOVEM.L  (SP)+,D2-D4/A0-A6     ;Restore registers
            RTS
```

```
;-------------------------------------------------------------------------
; Routine called to verify proper drive number
; Input: A0 - ptr to parameter block
;-------------------------------------------------------------------------
CkDrvNum
            MOVE.W  IOVDrvNum(A0),D0
            CMP.W   #DRV4,D0            ; check only for default drive for now
            BEQ     DiskRTS             ; return if OK
            ADDQ    #4,SP               ; else strip off return address
            MOVEQ   #NSDrvErr,D0        ; set bad drive number
            BRA     DiskDone            ; and exit driver


;*****************************************************************************
;
; Main entry point to driver
;
; Input - A0 = ptr to I/O parameter block
;         A1 = Address of Header buffer
;         A2 = Address of Data buffer
;         A3 = Address of Drive specific variables
;         A6 = Address of Driver variables
;         D2 = I/O block count
;         D3 = Starting block number (also saved in drive locals)
;
; Registers used
;         A0 = Hardware base address for reset lines
;         A4 = Hardware base address for data port
;         D0 = Scratch
;         D1 = Scratch
;         D2 = Scratch
;         D4 = Save of ptr to I/O parameter block
;
; Block Format used (532 bytes total):
;
;  Widget:
;         512 bytes data
;          12 bytes Mac file tags
;           7 bytes unused (=0)
;           1 byte checksum (8 bit XOR)
;
;  Profile:
;          12 bytes Mac file tags
;           7 bytes unused (=0)
;           1 byte checksum (8 bit XOR)
;         512 bytes data
;
;*****************************************************************************

ProDriver

        MOVEM.L D3-D4/A0-A4,-(SP)  ;SAVE REGS USED
        MOVE.L  HWBASE(A3),A4      ;Data port base address
        CMP.L   #VIABASE,A4        ;using builtin port?
        BNE.S   @1                 ;skip if not
        MOVEQ   #1,D0              ;else indicate parallel port is busy
```

```
        TRAPTO  _DiskSync

@1      MOVE.L  A0,D4               ;save param block ptr
        MOVE.L  HWRESET(A3),A0      ;set reset line base address
        CLR     ERROR(A3)           ;zero the error return code
        CLR     CNTRESETS(A3)       ; and the reset counter

        MOVE.B   COMMAND(A3),D0           ;Get command
        MOVE.W   START_STATE(D0),STATE(A3) ;Get initial starting state

LP1     MOVE    STATE(A3),D0        ;PICK UP STATE TABLE OFFSET
        MOVE    STATE_TABLE(D0),D0
        JSR     STATE_TABLE(D0)     ;CALL ROUTINE, WHICH MUST PRESERVE D4-D7,A2-A6,
        TST     D0                  ; AND RETURN WITH D0 <> 0 IFF CALL ANOTHER STATE
        BNE.S   LP1                 ;CALL ANOTHER ROUTINE IF D0 <> 0

;Exit the driver's state machine

EXIT_STATE
        CMP.L   #VIABASE,A4         ;using builtin port?
        BNE.S   @9                  ;skip if not
        MOVEQ   #0,D0               ;else mark parallel port as not busy
        TRAPTO  _DiskSync
@9      MOVEM.L (SP)+,D3-D4/A0-A4   ;RESTORE REGS
        RTS

;---------------------------------------------------------------------
; PROFILE/WIDGET STATE MACHINE
;---------------------------------------------------------------------

STATE_TABLE

;-------------------------------------------------------------------------------

START_STATE                         ;Starting state depending upon command
        .WORD   INIT_STATE-STATE_TABLE  ;Initialize Controller
        .WORD   IO_STATE-STATE_TABLE    ;I/O

;-------------------------------------------------------------------------------

INIT_STATE                          ;start of states to initialize controller
        .WORD   S80-STATE_TABLE         ;Initialize controller

;-------------------------------------------------------------------------------

IO_STATE                        ;Start of states for I/O
        .WORD   S0-STATE_TABLE  ;starting point for new I/O request

;-------------------------------------------------------------------------------

NEW_CMD                         ;USE SINGLE-BLOCK COMMAND
        .WORD   S1-STATE_TABLE          ;do handshake
        .WORD   S2-STATE_TABLE          ;do 2nd handshake; poll for 1ms -
                                        ;                  wait for int if longer
        .WORD   S3-STATE_TABLE          ;send command to disk
        .WORD   S1A-STATE_TABLE ;CONTINUE READING
```

```
        .WORD   S200-STATE_TABLE        ;do 2nd handshake, always wait for interrupt
        .WORD   S6-STATE_TABLE          ;read profile status
        .WORD   S7-STATE_TABLE          ;read data

WRT     .WORD   S1A-STATE_TABLE ;CONTINUE WRITING
        .WORD   S8-STATE_TABLE          ;do 2nd handshake and compute checksum
        .WORD   S10-STATE_TABLE         ;write data
        .WORD   S1A-STATE_TABLE         ;do handshake
        .WORD   S200-STATE_TABLE        ;do 2nd handshake, always wait for interrupt
        .WORD   S6-STATE_TABLE          ;read profile status
        .WORD   S13-STATE_TABLE         ;return, or start re-read to verify

;        .WORD   S1-STATE_TABLE  ;READ-BACK TO VERIFY AFTER WRITE
;        .WORD   S2-STATE_TABLE          ;do 2nd handshake; poll for 1ms -
                                        ;                  wait for int if longer
;        .WORD   S3-STATE_TABLE          ;send read command to disk
;        .WORD   S1A-STATE_TABLE         ;do handshake
;        .WORD   S200-STATE_TABLE        ;do 2nd handshake, always wait for interrupt
;        .WORD   S6-STATE_TABLE          ;read profile status
;        .WORD   S20-STATE_TABLE         ;compute checksum on data

HS      .WORD   S1-STATE_TABLE  ;EXTRA HANDSHAKE TO UPDATE SPARE TABLE
        .WORD   S2-STATE_TABLE          ;do 2nd handshake; poll for 1ms -
                                        ;                  wait for int if longer
        .WORD   S30-STATE_TABLE         ;send illegal command
        .WORD   S1A-STATE_TABLE         ;do handshake
        .WORD   S2-STATE_TABLE          ; send 55 regardless of response
        .WORD   S31-STATE_TABLE         ;return

BDR     .WORD   S40-STATE_TABLE ;BAD RESPONSE FROM HANDSHAKE
        .WORD   S1-STATE_TABLE          ;continue if widget - do handshake
        .WORD   S2-STATE_TABLE          ;do 2nd handshake; poll for 1ms -
                                        ;                  wait for int if longer
        .WORD   S41-STATE_TABLE         ;send 'read status' command
        .WORD   S1A-STATE_TABLE         ;do handshake
        .WORD   S2-STATE_TABLE          ;do 2nd handshake; poll for 1ms -
                                        ;                  wait for int if longer
        .WORD   S42-STATE_TABLE         ;read status - return bad response
;-------------------------------------------------------------------------------
MULTI_CMD                       ;USE MULTI-BLOCK COMMAND
        .WORD   S1-STATE_TABLE          ;do handshake
        .WORD   S2-STATE_TABLE          ;do 2nd handshake; poll for 1ms -
                                        ;                  wait for int if longer
        .WORD   S50-STATE_TABLE         ;send command to disk

RD_NEXT                         ;READ NEXT BLOCK IN MULTI-BLOCK COMMAND
        .WORD   S1A-STATE_TABLE         ;do handshake
        .WORD   S200-STATE_TABLE        ;do 2nd handshake, always wait for interrupt
        .WORD   S6-STATE_TABLE          ;read profile status
        .WORD   S51-STATE_TABLE         ;read data - loop to RD_NEXT for more blks
        .WORD   S1A-STATE_TABLE         ;Do handskake to free device
        .WORD   S2A-STATE_TABLE         ;do 2nd handshake; poll for 1ms -
                                        ;                  wait for int if longer
        .WORD   S52-STATE_TABLE         ;If needed start new multi_block request

WRT_NEXT                        ;WRITE NEXT BLOCK IN MULTI-BLOCK COMMAND
```

```
        .WORD   S10A-STATE_TABLE        ;write data
        .WORD   S1A-STATE_TABLE         ;Do first handshake
        .WORD   S53-STATE_TABLE         ;loop to WRT_NEXT for more blocks
        .WORD   S1A-STATE_TABLE         ;Do first handshake
        .WORD   S2-STATE_TABLE          ;do 2nd handshake; poll for 1ms -
                                        ;                  wait for int if longer
WRT_STATUS
        .WORD   S6-STATE_TABLE          ;Read status
        .WORD   S1-STATE_TABLE          ;Do handskake to free device
        .WORD   S2A-STATE_TABLE         ;do 2nd handshake; poll for 1ms -
                                        ;                  wait for int if longer
        .WORD   S52-STATE_TABLE         ;If needed start new multi_block request

        ;-------------------------------------------------------------
        ;INITIALIZE VARIABLES FOR FIRST HANDSHAKE
        ;  return: continue at next state

S0      MOVEQ   #1,D0                   ; advance to next state immediately
        TST.B   DRIVETYPE(A3)           ; Controller support system commands?
        BEQ.S   @1                      ; No - Go issue single block command
        MOVE.W  #MULTI_CMD-STATE_TABLE,STATE(A3) ; Next state for multi-block
        RTS                                      ;  command
@1      ADDQ    #2,STATE(A3)            ; advance to next state
        RTS

        ;-------------------------------------------------------------
        ; ASSERT "CMD" AND WAIT FOR "BSY"
        ;   return: continue at next state now, or wait for interrupt first
S1      MOVE.B  #1,ER_HS(A3)            ; Expected response
S1A     ADDQ    #2,STATE(A3)            ; advance to next state

;        ANDI.B  #$FE,PCR(A4)            ; interrupt on falling edge
        MOVE.B  #$02,IFR(A4)            ; clear pending ints

        ORI.B   #$08,ORB(A4)            ; set dir = in
        ANDI.B  #$EF,ORB(A4)            ; set cmd=true
        CLR.B   DDRA(A4)                ; set port A bits to input

;-------------------------------------------------------------------------------
        BSR     WAIT_BUSY               ;poll until busy
        TST     D0
        BEQ.S   GETRSP                  ;skip if OK
        MOVE.B  #TIME_ERR,ERROR(A3)     ;else set timeout error
        MOVEQ   #0,D0                   ;return now
        RTS                             ;and exit
;-------------------------------------------------------------------------------
;        MOVE    #RSPTIME,D0             ; set response timeout to about 1 ms
;WFB1    BTST    #1,IFR(A4)              ; wait for busy
;        BNE.S   GETRSP                  ; skip if OK
;        DBF     D0,WFB1                 ; else loop until timeout
;        MOVE    #WAIT_INT,ERROR(A3)     ; wait for interrupt -- parking head now
;        MOVE.B  #$FF,T2CH(A4)           ;START TIMER FOR DISCON ERROR CHECK
;        MOVEQ   #0,D0
;        RTS
;-------------------------------------------------------------------------------
```

```
GETRSP
        MOVE.B  #$02,IFR(A4)            ; clear interrupt flag
        MOVEQ   #1,D0                   ; continue at next state
        RTS


        ;-------------------------------------------------------------
        ;GET RESPONSE, WAIT FOR BSY FOR 1MS THEN RESORT TO WAIT FOR INTERRUPT
        ;  return: continue at next state or BDR state after interrupt
S2A     MOVE.B  #$69,D2                 ;Respond with free device reply
        BRA.S   S2_GO
S2      MOVE.B  #$55,D2                 ;Respond with standard reply
S2_GO   BSR.S   RESPOND

;-------------------------------------------------------------------------------
        BSR     WAIT_NOTBUSY            ;poll until not busy
        TST     D0
        BEQ.S   @2                      ;skip if OK
        MOVE.B  #TIME_ERR,ERROR(A3)     ;else set timeout error
        MOVEQ   #0,D0                   ;return now
        RTS                             ;and exit
;-------------------------------------------------------------------------------
;       MOVE    #RSPTIME,D0             ;Response timeout of 1ms
;@1     BTST    #1,IFR(A4)              ;Wait for not busy
;       BNE.S   @2                      ;Skip if OK
;       DBF     D0,@1                   ;Else loop till timeout
;       MOVE    #WAIT_INT,ERROR(A3)     ;Wait for interrupt
;       MOVE.B  #$FF,T2CH(A4)           ;START TIMER FOR DISCON ERROR CHECK
;       MOVEQ   #0,D0
;       RTS
;-------------------------------------------------------------------------------

@2      ;Controller not busy
        MOVE.B  #$02,IFR(A4)            ;Clear interrupt flag
        MOVEQ   #1,D0                   ;Continue at next state
        RTS


        ;-------------------------------------------------------------
        ;GET RESPONSE, ALWAYS WAIT FOR INTERRUPT
        ;  return: continue at next state or BDR state after interrupt
S200
;-------------------------------------------------------------------------------
        BRA.S   S2                      ;just poll for now
;-------------------------------------------------------------------------------
;       MOVE.B  #$55,D2                 ;Respond with standard reply
;       BSR.S   RESPOND
;       MOVE    #WAIT_INT,ERROR(A3)     ;Wait for interrupt
;       MOVE.B  #$FF,T2CH(A4)           ;START TIMER FOR DISCON ERROR CHECK
;       MOVEQ   #0,D0
;       RTS
;-------------------------------------------------------------------------------

        ;
        ;RESPOND TO PROFILE HANDSHAKE -- SUBROUTINE USED BY S2, S8, S200
        ;
```

```
        ;  Input:  D2 = Reply to be sent to controller if response OK
        ;  Output: D1 = response from controller
        ;          D2 = Final reply sent to controller
        ;  UPDATES STATE FOR ERROR OR "NEXT" STATE.
        ;

RESPOND
;        ORI.B   #$01,PCR(A4)            ; restore to interrupt on rising edge
        MOVE.B  PORTA(A4),D1            ; get response in D1
        CMP.B   ER_HS(A3),D1            ; did drive return state requested ?
        BEQ.S   RSPOK                   ; skip if yes
        TST.B   ER_HS(A3)
        BPL.S   BAD_RSP                 ; (0 IS WILD CARD ON INPUT
RSPOK   ADDQ    #2,STATE(A3)            ; advance to next state
SNDR1   ANDI.B  #$E7,ORB(A4)            ; set dir=out, cmd=true
        MOVE.B  #$FF,DDRA(A4)           ; set port A bits to output
        MOVE.B  D2,PORTA(A4)            ; send reply w/o handshake
        MOVE.B  #$02,IFR(A4)            ; clear interrupt flag
        ORI.B   #$10,ORB(A4)            ; set cmd=false
        RTS
BAD_RSP MOVE    #BDR-STATE_TABLE,STATE(A3);NEXT STATE - FLAGS BAD RESP ERROR
        MOVE.B  #0,D2                   ; Negative reply for Profile/Seagate
        TST.B   DRIVETYPE(A3)           ; Widget?
        BEQ.S   SNDR1
        MOVE.B  #$69,D2                 ; Negative reply for Widget
        BRA.S   SNDR1                   ; and go send reply



        ;TABLE OF INTERLEAVE REMAPPINGS FOR LOW 4 BITS OF BLOCK NUMBER
        ;9:1 INTERLEAVE ON TOP OF 5:1 FOR PROFILE OR SEAGATE (10 MB)
INTITAB .BYTE 0,5,10,15,4,9,14,3,8,13,2,7,12,1,6,11

        ;-------------------------------------------------------------
        ; SEND COMMAND BYTES OUT AND INITIALIZE VARIABLES FOR NEXT HANDSHAKE
        ;  return: continue at next state (read), continue at WRT state,
        ;          or return with parity error

S3      MOVEM.L A1,-(SP)                ;save regs
;-------------------------------------------------------------------------
;        ANDI.B  #$DF,ORB(A0)            ;CLEAR PARITY
;        ORI.B   #$20,ORB(A0)
;        MOVE.B  #$08,IFR(A4)
;-------------------------------------------------------------------------

        LEA     CMD_BUFFER(A3),A1       ; INITIALIZE REGS NEEDED
        MOVE.B  (A1)+,ORA(A4)           ; send a command byte
        MOVE.B  (A1)+,ORA(A4)           ; send a command byte
        MOVE.B  (A1)+,ORA(A4)           ; send a command byte

        ; do interleave remapping
        MOVE.B  (A1)+,D0                ;get low byte of block number
        MOVEQ   #$0F,D1                 ;DO THE REMAPPING
        AND     D0,D1                   ;MASK LOW 4 BITS
        AND.B   #$F0,D0                 ;MASK HIGH 4 BITS
        ADD.B   INTITAB(D1),D0          ;Add in remapped low 4 bits
```

```
        MOVE.B  D0,ORA(A4)              ; send a command byte
        MOVE.B  (A1)+,ORA(A4)           ; send a command byte
        MOVE.B  (A1)+,ORA(A4)           ; send a command byte
        TST.B   CMD_BUFFER(A3)          ;Write command?
        BNE.S   @9                      ;Skip if yes
        ADDQ    #2,STATE(A3)            ;Continue with read state next
        MOVE.B  #2,ER_HS(A3)            ;Expected response
        BRA.S   S3FINI                  ;Go finish state

@9      MOVE.W  #WRT-STATE_TABLE,STATE(A3) ;Go to write state next
        MOVE.B  #3,ER_HS(A3)               ;Expected response

S3FINI  ;finish off sending request
        MOVEQ   #1,D0                   ;Advance to next state now
;-----------------------------------------------------------------
;        BTST    #3,IFR(A4)              ;PARITY ERROR?
;        BEQ.S   @9                      ;Skip if no
;        MOVE    #PRTY_ERR,ERROR(A3)
;        MOVEQ   #0,D0                   ;RETURN NOW
;-----------------------------------------------------------------

@9      ORI.B   #$18,ORB(A4)            ;reset dir=in ; cmd=false
        CLR.B   DDRA(A4)                ;set port A bits to input
        MOVEM.L (SP)+,A1                ;restore regs
        RTS


RD_STATUS
        ;---------------------------------------------------------------
        ; Read the status and check parity
        ; Output - status bytes in STATUS(A3)
        ;        - A1 = STATUS(A3)
        ;        - D0 = 0 if status valid
        ;             = 1 if parity error reading status - status invalid
        ;
        MOVEM.L A1,-(SP)
        CLR.B   DDRA(A4)        ;SET PORT A TO INPUT (=0)
        ORI.B   #$18,ORB(A4)    ;SET DIR = IN

;-------------------------------------------------------
;        ANDI.B  #$DF,ORB(A0)    ;CLEAR PARITY
;        ORI.B   #$20,ORB(A0)
;        MOVE.B  #$08,IFR(A4)
;-------------------------------------------------------

        LEA     STATUS(A3),A1   ;GRAB ADDRESS OF ESTAT
        MOVE.B  IRA(A4),(A1)    ;READ 4 ERROR STATUS BYTES FROM DISK
        MOVE.B  IRA(A4),1(A1)
        MOVE.B  IRA(A4),2(A1)
        MOVE.B  IRA(A4),3(A1)

;-------------------------------------------------------
;        BTST    #3,IFR(A4)      ;PARITY ERROR?
;        BNE.S   @1
;-------------------------------------------------------
```

```
        MOVEQ   #0,D0           ;Valid status
        BRA.S   @2
@1      MOVEQ   #1,D0           ;Invalid status
@2      MOVEM.L (SP)+,A1        ;restore reg
        RTS


        ;-------------------------------------------------------------
        ;READ DISK STATUS
        ;  return:  continue at next state, or return with parity err or hard err

S6      BSR     RD_STATUS       ;Read status
        TST     D0              ;Status valid?
        BNE.S   @10
        CMPI.B  #$09,STATUS(A3)
        BEQ.S   @3              ;JUST CONTINUE IF GOT CRC ERROR ON READ
        MOVE.L  #$C140C000,D0
        AND.L   STATUS(A3),D0   ;SEE IF FATAL ERROR PRESENT
        BEQ.S   @3
        MOVE    #HD_ERR,ERROR(A3)  ; YES
        MOVEQ   #0,D0
        RTS
@10     MOVE    #PRTY_ERR,ERROR(A3) ;status not valid
        MOVEQ   #0,D0
        RTS
@3      ADDQ    #2,STATE(A3)        ; advance to next state
        MOVEQ   #1,D0               ; do it now
        RTS

        ;RDHDR     READ HEADER FROM PROFILE
        ;          A0 = I/O address
        ;          A1 = File tag ADDRESS
        ;          D0,D2 = SCRATCH
        ;          D1 = CHECKSUM, UPDATED WITH THIS DATA

RDHDR   MOVE.L  A0,-(SP)            ;save reg
        MOVEQ   #2,D2               ;read 12 bytes of file tags into buffer
        LEA     IRA(A4),A0          ;setup read address
@1      MOVE.B  (A0),D0             ;get 1st byte
        EOR.B   D0,D1               ;add to checksum
        MOVE.B  D0,(A1)+            ;move to buffer
        MOVE.B  (A0),D0             ;get 2nd byte
        EOR.B   D0,D1
        MOVE.B  D0,(A1)+
        MOVE.B  (A0),D0             ;get 3rd byte
        EOR.B   D0,D1
        MOVE.B  D0,(A1)+
        MOVE.B  (A0),D0             ;get 4th byte
        EOR.B   D0,D1
        MOVE.B  D0,(A1)+
        DBF     D2,@1

        MOVEQ   #1,D2               ;get remaining 8 bytes of header
@2      MOVE.B  (A0),D0
        EOR.B   D0,D1
```

```
        NOP                             ;needed for timing
        MOVE.B  (A0),D0
        EOR.B   D0,D1
        NOP
        MOVE.B  (A0),D0
        EOR.B   D0,D1
        NOP
        MOVE.B  (A0),D0
        EOR.B   D0,D1
        DBF     D2,@2

        MOVE.L  (SP)+,A0                ;restore reg and exit
        RTS


        ;RDDATA    READ DATA FROM PROFILE/WIDGET, AND UPDATE CHECKSUM
        ;          A0 = I/O ADDRESS,
        ;          A2 = DATA ADDRESS(CLOBBERED ON RETURN)
        ;          D0,D2 = SCRATCH
        ;          D1 = CHECKSUM, UPDATED WITH THIS DATA
        ;
        ;  NOTE:  OPTIMAL READ RATE HAS 14-21 CPU CYCLES BETWEEN BYTES (INCLUDING
        ;            THE READ OPERATION ITSELF).  ANY FEWER CYCLES, AND THE PULSE
        ;            HANDSHAKE IS NOT GUARANTEED TO BEAT THE NEXT READ.

RDDATA  MOVE.L  A0,-(SP)        ;save reg
        MOVEQ   #63,D2          ;DO FAST READ, BY PROCESSING 8 AT A TIME
        LEA     IRA(A4),A0      ;setup read address
@3      MOVE.B  (A0),D0
        EOR.B   D0,D1           ;EXCLUSIVE-OR
        MOVE.B  D0,(A2)+        ;SAVE DATA IN BUFFER
        MOVE.B  (A0),D0
        EOR.B   D0,D1           ;EXCLUSIVE-OR
        MOVE.B  D0,(A2)+        ;SAVE DATA IN BUFFER
        MOVE.B  (A0),D0
        EOR.B   D0,D1           ;EXCLUSIVE-OR
        MOVE.B  D0,(A2)+        ;SAVE DATA IN BUFFER
        MOVE.B  (A0),D0
        EOR.B   D0,D1           ;EXCLUSIVE-OR
        MOVE.B  D0,(A2)+        ;SAVE DATA IN BUFFER
        MOVE.B  (A0),D0
        EOR.B   D0,D1           ;EXCLUSIVE-OR
        MOVE.B  D0,(A2)+        ;SAVE DATA IN BUFFER
        MOVE.B  (A0),D0
        EOR.B   D0,D1           ;EXCLUSIVE-OR
        MOVE.B  D0,(A2)+        ;SAVE DATA IN BUFFER
        MOVE.B  (A0),D0
        EOR.B   D0,D1           ;EXCLUSIVE-OR
        MOVE.B  D0,(A2)+        ;SAVE DATA IN BUFFER
        MOVE.B  (A0),D0
        EOR.B   D0,D1           ;EXCLUSIVE-OR
        MOVE.B  D0,(A2)+        ;SAVE DATA IN BUFFER
        DBF     D2,@3           ;REPEAT
        MOVE.L  (SP)+,A0        ;restore
        RTS
```

```
        ;-------------------------------------------------------------
        ;READ DATA
        ;  Used by devices that must issue single-block commands
        ;  return: continue in HS state, or return without error or with parity
        ;          or checksum error

S7      MOVEQ   #0,D1           ;INIT CHECKSUM VALUE
        BSR.S   RDHDR           ;READ HEADER FIRST
        BSR.S   RDDATA          ;THEN DATA

        ;Error conditions
        ; Checksum error, parity error, crc error - return immediately with cs_err
        ; Sparing occurred - if profile then do extra sparing handshake

        MOVEQ   #0,D0
        TST.B   D1              ;NOW SEE IF CHECKSUM IS ZERO
        BEQ.S   CS_OK
CER     MOVE    #CS_ERR,ERROR(A3) ;NON-ZERO CHECKSUM ERROR
        RTS                     ;RETURN WITH D0=0

CS_OK

;        BTST    #3,IFR(A4)
;        BNE.S   CER                ;PARITY ERROR?

        CMPI.B  #$09,STATUS(A3)
        BEQ.S   CER                ;now RETURN WITH CHECKSUM ERROR IF GOT CRC ERROR

        ;Issue new command to continue request (sect_left > 0)
        ;Exit (sect_left = 0)

@20     BSR.S   ADJ_HDR            ;call routine for header adjustment
        BSR.S   ADJ_NEXTBLK        ;and setup for next block
        ADDQ.L  #1,SECTOR(A3)      ;update logical block ptr
        SUBQ.L  #1,SECT_LEFT(A3)   ;decr sector count
        BEQ.S   @40                ;Skip if no more sectors
        MOVE.W  #NEW_CMD-STATE_TABLE,STATE(A3) ;Issue new command
        MOVEQ   #1,D0              ;Advance to next state now
        RTS

        ;Test sparing on last sector
@40     BTST    #2,STATUS+1(A3)    ;Sparing occur?
        BNE.S   @41                ;skip if yes
        MOVEQ   #0,D0              ;return immediately
        RTS
@41     MOVE    #HS-STATE_TABLE,STATE(A3) ;DO ONE MORE HANDSHAKE IF SPARING
        MOVEQ   #1,D0              ;Do it now
        RTS

;Adjust file tags as necessary

ADJ_HDR
        TST.B   IOCOMMAND(A3)      ;read command?
        BEQ.S   @1
```

```
        ;Compute new tag for write
        MOVE.W  FSBLKSDONE(A3),D0     ;get # of blocks done
        ADD.W   FIRSTFSEQ(A3),D0      ;compute relative block #
        MOVE.W  D0,TAGDATA+8          ;save as part of file tag
        MOVE.L  TIME,TAGDATA+10       ;and timestamp it

        ;Reset tag buffer ptr if necessary
@1      MOVE    #TagData+2,A1         ;assume use of default tag buffer area
        TST.L   TAGBUFPTR(A6)         ;separate tag buffer exists?
        BEQ.S   @2                    ;skip if not
        MOVE.L  TAGBUFPTR(A6),A1      ;else use it

@2      RTS


;Update position and bytes read fields

ADJ_NEXTBLK
        MOVE.L  A0,-(SP)              ;save reset port ptr
        MOVE.L  D4,A0                 ;get I/O block ptr
        MOVEQ   #2,D0
        ASL.L   #8,D0                 ;set D0 = 512
        ADD.L   D0,IONumDone(A0)      ;update # of bytes read
        ADD     #1,FSBLKSDONE(A3)     ;update # of blocks done

        MOVE.L  DCE(A6),A0            ;get ptr to DCE
        ADD.L   D0,DctlPosition(A0)   ;update byte position
        MOVE.L  (SP)+,A0              ;restore ptr
        RTS


C_SUM   ;Precompute the checksum before writing the sector
        MOVEM.L A1/A2,-(SP)     ;save header/data ptrs
        MOVEQ   #0,D1           ;ACCUMULATE CHECKSUM IN D1, STARTING WITH HEADER
        MOVEQ   #2,D2           ;REPEAT 4 BYTES 3 TIMES FOR 12 byte file tags
@1      MOVE.L  (A1)+,D0
        EOR.L   D0,D1           ;FOUR-BYTE CHECKSUM UPDATE FOR HEADER
        DBF     D2,@1           ;REPEAT

        MOVEQ   #127,D2         ;NEXT INCLUDE 512 BYTES OF DATA
        MOVE    A2,D0           ;CHECK FOR ODD ADDRESS
        AND     #1,D0
        BEQ.S   @4              ;SKIP IF EVEN
        MOVE.B  511(A2),D0      ;ODD, SO INCLUDE LAST 1 AND FIRST 3 BYTES NOW
        EOR.B   D0,D1
        MOVE.B  (A2)+,D0
        EOR.B   D0,D1
        MOVE.W  (A2)+,D0
        EOR.W   D0,D1
        SUBQ    #1,D2           ;AND GO THRU LOOP 4 AT A TIME ONE FEWER ITERATIONS
@4      MOVE.L  (A2)+,D0
        EOR.L   D0,D1
        DBF     D2,@4           ;REPEAT
        MOVE.W  D1,D0           ;XOR 4 CHECKSUM BYTES TOGETHER
        SWAP    D1
```

```
        EOR.W   D0,D1
        MOVE.W  D1,CHECKSUM(A3)
        EOR.B   D1,CHECKSUM(A3) ;RETURN THE CHECKSUM BYTE
        MOVEM.L (SP)+,A1/A2     ;restore ptrs
        RTS


        ;---------------------------------------------------------------
        ;GET RESPONSE, & PRE-COMPUTE CHECKSUM
        ;  return: continue at next state or BDR state, now or after interrupt

S8      MOVE.B  #$55,D2                 ;Respond with standard reply
        BSR.S   RESPOND
        CMPI.B  #$55,D2                 ;Did reply get changed?
        BNE.S   @5                      ;YES, bad response, wait for interrupt
        BSR.S   C_SUM                   ;Precompute the sector's checksum
;-------------------------------------------------------------------------------
        MOVE.B  #1,CSUM_VALID(A3)       ;mark checksum as valid
        BRA.S   @6
@5      MOVE.B  #0,CSUM_VALID(A3)       ;mark checksum as invalid
@6      BSR     WAIT_NOTBUSY            ;poll until not busy
        TST     D0
        BEQ.S   @2                      ;skip if OK
        MOVE.B  #TIME_ERR,ERROR(A3)     ;else set timeout error
        MOVEQ   #0,D0                   ;return now
        RTS
                                ;and exit
@2      ;Controller not busy
        MOVE.B  #$02,IFR(A4)            ;Clear interrupt flag
        MOVEQ   #1,D0                   ;Continue at next state
        RTS
;-------------------------------------------------------------------------------

;-------------------------------------------------------------------------------
;       BTST    #1,IFR(A4)              ; interrupt pending yet?
;       BEQ.S   @5                      ; no - wait for the interrupt
;       MOVE.B  #1,CSUM_VALID(A3)       ; precomputed checksum is valid
;       MOVE.B  #$02,IFR(A4)            ; clear interrupt flag
;       MOVEQ   #1,D0                   ; do it now
;       RTS
;@5     MOVE.B  #0,CSUM_VALID(A3)       ; precomputed checksum is invalid
;       MOVE    #WAIT_INT,ERROR(A3)     ; wait for interrupt
;       MOVE.B  #$FF,T2CH(A4)           ;START TIMER FOR DISCON ERROR CHECK
;       MOVEQ   #0,D0
;       RTS
;-------------------------------------------------------------------------------

        ;---------------------------------------------------------------
        ; WRITE DATA - Header followed by User Data
        ;  return: continue at next state, or return with parity error

S10     TST.B   CSUM_VALID(A3)          ;precomputed checksum valid?
        BNE.S   @10                     ;skip if yes
        BSR     C_SUM                   ;compute the checksum
@10     MOVE.B  CHECKSUM(A3),D1         ;GET PREVIOUSLY-COMPUTED CHECKSUM
        ANDI.B  #$F7,ORB(A4)            ;SET DIR=OUT
```

```
        MOVE.B  #$FF,DDRA(A4)           ;SET PORT A BITS TO OUTPUT

;-------------------------------------------------------
;        ANDI.B  #$DF,ORB(A0)            ;CLEAR PARITY
;        ORI.B   #$20,ORB(A0)
;        MOVE.B  #$08,IFR(A4)
;-------------------------------------------------------

        BSR     WRHDR                   ;write header
        BSR.S   WRDATA                  ;write data
        BRA     FINI_WRITE              ;Go complete write

        ;-------------------------------------------------------------
        ; WRITE DATA - User Data followed by Header
        ;  return: continue at next state, or return with parity error

S10A    ANDI.B  #$F7,ORB(A4)            ;SET DIR=OUT
        MOVE.B  #$FF,DDRA(A4)           ;SET PORT A BITS TO OUTPUT

;        ANDI.B  #$DF,ORB(A0)            ;CLEAR PARITY
;        ORI.B   #$20,ORB(A0)
;        MOVE.B  #$08,IFR(A4)

        BSR.S   WR_WDATA                ;write data
        BSR     WR_WHDR                 ;write header

FINI_WRITE  ;Finish Write

;-------------------------------------------------------------------------
;        BTST    #3,IFR(A4)              ;CHECK PARITY ERROR BIT
;        BEQ.S   @1                      ;SKIP IF NO ERROR
;        CLR.B   DDRA(A4)                ;SET PORT A TO INPUT
;        ORI.B   #$08,ORB(A4)            ;SET DIR=IN (ALSO CLEARS PARITY ERR FLAG)
;        MOVE    #PRTY_ERR,ERROR(A3)
;        MOVEQ   #0,D0                   ;RETURN NOW
;        RTS
;-------------------------------------------------------------------------

@1      CLR.B   DDRA(A4)            ;SET PORT A TO INPUT
        ORI.B   #$08,ORB(A4)        ;SET DIR=IN (ALSO CLEARS PARITY ERR FLAG)
        SUBQ.L  #1,SECT_LEFT(A3)    ;decr sector count
        MOVE.B  #6,ER_HS(A3)        ;EXPECT 6 IN NEXT HANDSHAKE
        ADDQ    #2,STATE(A3)        ; advance to next state
        MOVEQ   #1,D0               ; CONTINUE
        RTS


        ;WRDATA   WRITE DATA BUFFER TO PROFILE
        ;A0 = HARDWARE OUTPUT ADDRESS
        ;A2 = DATA BUFFER (CLOBBERED ON RETURN)
        ;D0 = SCRATCH
        ;
        ;  SEQUENTIAL WRITES MUST OPTIMALLY TAKE 14-21 CPU
        ;    CYCLES, INCLUDING THE WRITE INSTRUCTION ITSELF

WRDATA
```

```
        MOVE.L  A0,-(SP)        ;save reg
        MOVEQ   #127,D0         ;WRITE 4 BYTES, 128 TIMES (512 BYTES TOTAL)
        LEA     ORA(A4),A0      ;setup port address
@1      MOVE.B  (A2)+,(A0)      ;WRITE 1ST BYTE OUT
        NOP
        MOVE.B  (A2)+,(A0)      ;WRITE 2ND BYTE OUT
        NOP
        MOVE.B  (A2)+,(A0)      ;WRITE 3RD BYTE OUT
        NOP
        MOVE.B  (A2)+,(A0)      ;WRITE 4TH BYTE OUT
        DBF     D0,@1           ;REPEAT
        MOVE.L  (SP)+,A0        ;restore reg and exit
        RTS

        ;WR_WDATA   WRITE DATA BUFFER TO WIDGET - COMPUTE CHECKSUM ON FLY
        ;Input -
        ;       A0 = DATA BUFFER (CLOBBERED ON RETURN)
        ;       A2 = HARDWARE OUTPUT ADDRESS
        ;       D0 = SCRATCH
        ;Output -
        ;       D1 = Computed checksum
        ;
        ;  SEQUENTIAL WRITES MUST OPTIMALLY TAKE 14-21 CPU
        ;    CYCLES, INCLUDING THE WRITE INSTRUCTION ITSELF

WR_WDATA
        MOVE.L  A0,-(SP)        ;save reg
        LEA     ORA(A4),A0      ;setup write address
        MOVEQ   #0,D1           ;Start of checksum
        MOVEQ   #127,D0         ;WRITE 4 BYTES, 128 TIMES (512 BYTES TOTAL)
@1      MOVE.B  (A2)+,D2        ;Get 1st byte
        EOR.B   D2,D1           ;Include in checksum
        MOVE.B  D2,(A0)         ;Write 1st byte
        MOVE.B  (A2)+,D2        ;Get 2nd byte
        EOR.B   D2,D1           ;Include in checksum
        MOVE.B  D2,(A0)         ;Write 2nd byte
        MOVE.B  (A2)+,D2        ;Get 3rd byte
        EOR.B   D2,D1           ;Include in checksum
        MOVE.B  D2,(A0)         ;Write 3rd byte
        MOVE.B  (A2)+,D2        ;Get 4th byte
        EOR.B   D2,D1           ;Include in checksum
        MOVE.B  D2,(A0)         ;Write 4th byte
        DBF     D0,@1           ;REPEAT
        MOVE.L  (SP)+,A0        ;restore
        RTS


        ;WRHDR    WRITE HEADER TO PROFILE
        ;A1 = FILE TAG BUFFER (A1 CLOBBERED ON RETURN),
        ;A0 = HARDWARE OUTPUT ADDRESS, D1 = CHECKSUM BYTE, D2 = SCRATCH

WRHDR
        MOVE.L   A0,-(SP)           ;save reg
        MOVEQ    #2,D2              ;first write file tags
        LEA      ORA(A4),A0         ;setup read address
@1      MOVE.B   (A1)+,(A0)         ;write 1st byte
```

```
        NOP                     ;for timing
        MOVE.B  (A1)+,(A0)      ;write 2nd byte
        NOP
        MOVE.B  (A1)+,(A0)      ;write 3rd byte
        NOP
        MOVE.B  (A1)+,(A0)      ;write 4th byte
        NOP
        DBF     D2,@1

        CLR     D0              ;write 7 bytes of 0
@2      MOVE.B  D0,(A0)         ;write it
        NOP                     ;need for timing
        MOVE.B  D0,(A0)
        NOP
        MOVE.B  D0,(A0)
        NOP
        MOVE.B  D0,(A0)
        NOP
        MOVE.B  D0,(A0)
        NOP
        MOVE.B  D0,(A0)
        NOP
        MOVE.B  D0,(A0)
        NOP

;  finally write out the checksum

        MOVE.B  CHECKSUM(A3),(A0)  ;write checksum as 20th header byte
        MOVE.L  (SP)+,A0           ;restore reg and exit
        RTS


        ;WR_WHDR    WRITE HEADER TO WIDGET - COMPUTE CHECKSUM ON FLY
        ; Input -
        ;       A0 = HARDWARE OUTPUT ADDRESS
        ;       A1 = FILE TAG BUFFER (A1 CLOBBERED ON RETURN),
        ;       D0 = SCRATCH
        ;       D1 = CHECKSUM BYTE
        ;       D2 = SCRATCH
WR_WHDR
        MOVE.L  A0,-(SP)        ;save reg
        MOVEQ   #2,D2           ;first write file tags
        LEA     ORA(A4),A0      ;setup read address
@1      MOVE.B  (A1)+,D0        ;get 1st byte
        EOR.B   D0,D1           ;add to checksum
        MOVE.B  D0,(A0)         ;write it
        MOVE.B  (A1)+,D0        ;get 2nd byte
        EOR.B   D0,D1           ;add to checksum
        MOVE.B  D0,(A0)         ;write it
        MOVE.B  (A1)+,D0        ;get 3rd byte
        EOR.B   D0,D1           ;add to checksum
        MOVE.B  D0,(A0)         ;write it
        MOVE.B  (A1)+,D0        ;get 4th byte
        EOR.B   D0,D1           ;add to checksum
        MOVE.B  D0,(A0)         ;write it
        DBF     D2,@1
```

```
        CLR     D0                  ;write 7 bytes of 0
@2      MOVE.B  D0,(A0)             ;write it
        NOP                         ;need for timing
        MOVE.B  D0,(A0)
        NOP
        MOVE.B  D0,(A0)
        NOP
        MOVE.B  D0,(A0)
        NOP
        MOVE.B  D0,(A0)
        NOP
        MOVE.B  D0,(A0)
        NOP
        MOVE.B  D0,(A0)
        NOP

;  finally write out the checksum

        MOVE.B  D1,(A0)             ;write checksum as 20th header byte
        MOVE.L  (SP)+,A0            ;restore reg and exit
        RTS


        ;-------------------------------------------------------------
        ;WRITE SUCCEEDED, SEE IF NEED TO VERIFY
        ;  Used only by devices that do not support multi-block commands
        ;  return: continue at next state (read) or at HS state, or return successful

        ;VERIFY NOT CURRENTLY SUPPORTED

S13
;-----------------------------------------------------------------------
;       TST.B   V_FLAG(A6)
;       BEQ.S   @20                 ;SKIP IF NO VERIFY NEEDED
;       ADDQ    #2,STATE(A3)        ; advance to next state
;       CLR.B   CMD_BUFFER(A3)      ; CHANGE COMMAND TO READ
;       MOVEQ   #1,D0               ; CONTINUE
;       RTS
;-----------------------------------------------------------------------

        ;Issue new command to continue request (sect_left ) 0)
        ;Exit (sect_left = 0)

@20     BSR.S   ADJ_HDR             ;call routine for header adjustment

        TST.B   INITRQST(A6)        ;doing initialize?
        BEQ.S   @25                 ;skip if not
        MOVE.L  A0,-(SP)            ;save reset ptr
        MOVE.L  D4,A0               ;get param blk ptr
        MOVE.L  IOBUFFER(A0),A2     ;else restore buffer ptr
        MOVE.L  (SP)+,A0            ;restore reset ptr
        BRA.S   @30                 ;and continue

@25     BSR.S   ADJ_NEXTBLK         ;and setup for next block
```

```
@30     ADDQ.L  #1,SECTOR(A3)       ;update logical block ptr
        TST.L   SECT_LEFT(A3)       ;more to transfer?
        BEQ.S   @40                 ;Skip if no more sectors
        MOVE.W  #NEW_CMD-STATE_TABLE,STATE(A3) ;Issue new command
        MOVEQ   #1,D0               ;Advance to next state now
        RTS

        ;Test sparing on last sector
@40     BTST    #2,STATUS+1(A3)     ;Sparing occur?
        BNE.S   @41                 ;skip if yes
        MOVEQ   #0,D0               ;return immediately
        RTS
@41     MOVE    #HS-STATE_TABLE,STATE(A3) ;DO ONE MORE HANDSHAKE IF SPARING
        MOVEQ   #1,D0               ;Do it now
        RTS

        ;-------------------------------------------------------------
        ;READ DATA TO COMPUTE CHECKSUM ONLY
        ;  return: continue in HS state, or return without error or with parity
        ;          or checksum error

S20     MOVEQ   #0,D1               ;INIT CHECKSUM VALUE
        TST.B   DRIVETYPE(A3)       ;WIDGET?
        BNE.S   @10                 ;SKIP IF YES

        ;Profile/Seagate
        BSR.S   RHDR                ;READ HEADER FIRST
        MOVE    #511,D2             ;READ DATA
@3      MOVE.B  (A0),D0
        EOR.B   D0,D1               ;EXCLUSIVE-OR
        DBF     D2,@3               ;REPEAT
        BRA.S   @20

        ;Widget
@10     MOVE    #511,D2             ;READ DATA FIRST
@11     MOVE.B  (A0),D0
        EOR.B   D0,D1               ;EXCLUSIVE-OR
        DBF     D2,@11              ;REPEAT
        BSR.S   RHDR                ;READ HEADER
@20     TST.B   D1                  ;NOW SEE IF CHECKSUM IS ZERO
        BEQ.S   CS_OK2
CER2    MOVE    #CS_ERR,ERROR(A3)   ;NON-ZERO CHECKSUM OR PARITY ERROR IN DATA
        MOVEQ   #0,D0
        RTS

CS_OK2  BTST    #3,IFR(A4)
        BNE.S   CER2            ;PARITY ERROR?
        CMPI.B  #$09,STATUS(A3)
        BEQ.S   CER2            ;now RETURN WITH CHECKSUM ERROR IF CRC READ ERR

        ;Issue new command to continue request (sect_left > 0)
        ;Exit (sect_left = 0)
@20     BSR.S   ADJ_HDR             ;call routine for header adjustment
        BSR.S   ADJ_NEXTBLK         ;and setup for next block
        ADDQ.L  #1,SECTOR(A3)       ;update logical block ptr
        SUBQ.L  #1,SECT_LEFT(A3)    ;decr sector count
```

```
        BEQ.S   @30                 ;Skip if no more sectors
        MOVE.B  #1,CMD_BUFFER(A3)   ; Change command back to write
        MOVE.W  #NEW_CMD-STATE_TABLE,STATE(A3) ;Issue new command
        MOVEQ   #1,D0               ;Advance to next state now
        RTS

@30     BTST    #2,STATUS+1(A3)     ;TEST "SPARING OCCURRED" BIT
        BNE.S   @40                 ;skip if set SET
        MOVEQ   #0,D0
        RTS                         ;EXIT

@40     MOVE    #HS-STATE_TABLE,STATE(A3) ;SET TO DO ONE MORE HANDSHAKE IF SPARING
        MOVEQ   #1,D0
        RTS


        ;RHDR    READ HEADER FROM PROFILE
        ;         A0 = HARDWARE READ ADDRESS
        ;         D0,D2= SCRATCH, D1 = CHECKSUM, UPDATED WITH THIS DATA
        ;         D3 = CHECKSUM-PRESENT FLAG BYTE (7TH BYTE OF HEADER) RETURNED

RHDR    MOVE.L  A0,-(SP)        ;save reg
        LEA     IRA(A4),A0      ;set read address
        MOVEQ   #19,D2          ;GET 20 BYTES
@2      MOVE.B  (A0),D0         ;GET BYTE FROM DISK
        EOR.B   D0,D1           ;INCLUDE IN RUNNING CHECKSUM
        DBF     D2,@2           ;REPEAT UNTIL DONE
        MOVE.L  (SP)+,A0        ;restore
        RTS

        ;---------------------------------------------------------------
        ;DO EXTRA HANDSHAKE WHEN SPARING OCCURRED, BY SENDING ILLEGAL COMMAND
        ;  return: continue at next state

S30     MOVE.B  #$FF,ORA(A4)            ; send ILLEGAL command byte
        ORI.B   #$18,ORB(A4)            ; reset dir=in
        CLR.B   DDRA(A4)                ;  and set port A bits to input
        MOVE.B  #$FF,ER_HS(A3)          ; MATCH ANYTHING ON NEXT HANDSHAKE
        ADDQ    #2,STATE(A3)            ; advance to next state
        MOVEQ   #1,D0                   ; CONTINUE
        RTS

        ;---------------------------------------------------------------
        ;EXIT AFTER EXTRA HANDSHAKE
        ;  return: return successful

S31     ORI.B   #$18,ORB(A4)            ; reset dir=in
        CLR.B   DDRA(A4)                ;  and set port A bits to input
        MOVEQ   #0,D0
        RTS                             ;EXIT

        ;---------------------------------------------------------------
        ;BAD RESPONSE FROM HANDSHAKE
        ;  return: return with bad-response error

S40     TST.B   DRIVETYPE(A3)           ; Profile or Seagate device?
```

```
        BEQ.S   @10

        ;Widget device
        TST.B   NESTED_BDR(A3)          ; Bad response while handling bad response?
        BNE.S   @10
        MOVE.B  #1,NESTED_BDR(A3)       ; Start processing bad response
        MOVE.L  #RDTIME,D0              ; Wait for controller not busy
@1      BTST    #1,IRB(A4)
        BNE.S   @2
        DBF     D0,@1
        BRA.S   @10                     ; Give up if timeout
@2      ADDQ    #2,STATE(A3)            ; Go to next state
        MOVEQ   #1,D0                   ; Do it now
        RTS

        ;Profile/Seagate or nested error
@10     CLR.B   NESTED_BDR(A3)          ; Nested error or not Widget - give up
        MOVE    #RESP_ERR,ERROR(A3)
        CLR.B   DDRA(A4)                ; IN
        ORI.B   #$18,ORB(A4)            ; dir = in, cmd false
        MOVEQ   #0,D0                   ; return
        RTS

        ;-----------------------------------------------------------------
        ;SEND "READ CONTROLLER ABORT STATUS" COMMAND

S41
;-----------------------------------------------------------------------
;       ANDI.B  #$DF,ORB(A0)            ;CLEAR PARITY
;       ORI.B   #$20,ORB(A0)
;       MOVE.B  #$08,IFR(A4)
;-----------------------------------------------------------------------

       MOVE.B  #$13,ORA(A4)             ;Diagnostic command
       NOP
       MOVE.B  #$01,ORA(A4)             ;Read status instruction
       NOP
       MOVE.B  #$05,ORA(A4)             ;Read state registers
       NOP
       MOVE.B  #$E6,ORA(A4)             ;Check byte
       ORI.B   #$18,ORB(A4)             ;reset dir=in ; cmd=false
       CLR.B   DDRA(A4)                 ;set port A bits to input

;-----------------------------------------------------------------------
;       BTST    #3,IFR(A4)              ;PARITY ERROR?
;       BNE.S   @1                      ;Skip if yes
;-----------------------------------------------------------------------

       ADDQ    #2,STATE(A3)             ;Go to next state
       MOVE.B  #$3,ER_HS(A3)            ;Expected response
       MOVEQ   #1,D0                    ;Advance to next state now
       RTS

@1     MOVE    #PRTY_ERR,ERROR(A3)
       MOVEQ   #0,D0                    ;RETURN NOW
       RTS
```

```
        ;-------------------------------------------------------------
        ;READ CONTROLLER'S STATUS

S42     MOVE.L  A1,-(SP)                ;save header address
        CLR.B   DDRA(A4)                ;SET PORT A TO INPUT (=0)
        ORI.B   #$18,ORB(A4)            ;SET DIR = IN
        LEA     STATUS(A3),A1           ;Address of error status field
        MOVEQ   #3,D0                   ;Read 4 status bytes
@1      MOVE.B  IRA(A4),(A1)+           ;Read a byte
        DBF     D0,@1
        MOVE.L  STATUS(A3),D0
        OR.L    D0,ACCSTAT(A3)          ;accumulate state register status

        ;reset controller
        ANDI.B  #$F7,ORB(A4)            ;Dir = out
        ANDI.B  #$7F,RESETC(A4)         ;Toggle reset line
        MOVEQ   #39,D0                  ;(wait 100 mic sec)
@2      DBF     D0,@2
        ORI.B   #$80,RESETC(A4)
        MOVEQ   #127,D0                 ;(wait 1MS)
@3      DBF     D0,@3
        ADDI.W  #1,CNTRESETS(A3)        ;Increment reset counter
        MOVE.B  #0,NESTED_BDR(A3)
        MOVE.L  #$180000,D0             ;Wait for not busy - about 16 seconds
@10     BTST    #1,IRB(A4)
        BNE.S   @11
        SUBQ.L  #1,D0
        BNE.S   @10
@11     CMPI.W  #16,CNTRESETS(A3)
        BGT.S   @12
        MOVE.W  #MULTI_CMD-STATE_TABLE,STATE(A3)
        MOVEQ   #1,D0
        BRA.S   @14

@12     MOVE    #RESP_ERR,ERROR(A3)
        CLR.B   DDRA(A4)                ; IN
        ORI.B   #$18,ORB(A4)            ; dir = in, cmd false
        MOVEQ   #0,D0                   ; return

@14     MOVE.L  (SP)+,A1                ;restore header address
        RTS

SEND_CMD ;Send Widget command
         ;Input
         ; CMD_BUF = command type, command, and any parameters
         ;Output
         ; D0 = result of issue = 0 OK ; =1 Parity error during send
         ; D1 = scratch

        MOVE.L  A1,-(SP)

;        ANDI.B  #$DF,ORB(A0)            ;CLEAR PARITY
;        ORI.B   #$20,ORB(A0)
;        MOVE.B  #$08,IFR(A4)
```

```
        LEA     CMD_BUF(A3),A1
        MOVE.B  CMD_BUF(A3),D0          ;Command type and length
        EXT     D0
        ANDI    #$000F,D0               ;Extract length
        SUBQ    #1,D0
        MOVEQ   #0,D1                   ;Start of checkbyte
@1      MOVE.B  (A1),ORA(A4)            ;Send byte
        ADD.B   (A1)+,D1                ;Add to checkbyte
        DBF     D0,@1
        EORI.B  #$FF,D1
        MOVE.B  D1,ORA(A4)              ;Check byte

;        BTST    #3,IFR(A4)              ;PARITY ERROR?
;        BNE.S   @2                      ;Skip if yes

        MOVEQ   #0,D0                   ;Flag as OK
        BRA.S   @3
@2      MOVEQ   #1,D0                   ;Flag as bad
@3      MOVE.L  (SP)+,A1                ;restore reg
        RTS

        ;---------------------------------------------------------------
        ; FOR DEVICES THAT SUPPORT MULTI-BLOCK COMMANDS
        ; SEND COMMAND BYTES OUT AND INITIALIZE VARIABLES FOR NEXT HANDSHAKE
        ;  return: continue at next state (rd_next or wrt_next),
        ;          or return with parity error

S50
;        CMPI.B  #2,CMD_BUFFER(A3)       ;Non-I/O request?
;        BGE     S50NONIO

        ;Send Multi-block I/O transfer request
        MOVE.L  SECT_LEFT(A3),D0        ;Transfer count
        CMPI.W  #MAXCNT,D0              ;Max multi-block count
        BLE.S   @1
        MOVE.W  #MAXCNT,D0
@1      MOVE.B  D0,CXFERCNT(A3)         ;Current sector count
        MOVE.B  #$26,CMD_BUF(A3)        ;command and length
        MOVE.B  CMD_BUFFER(A3),CMD_BUF+1(A3) ; Command
        MOVE.L  CMD_BUFFER(A3),CMD_BUF+2(A3) ; Command and sector
        MOVE.B  D0,CMD_BUF+2(A3)        ;block count over command
        BSR     SEND_CMD                ;Issue Widget command
        TST     D0                      ;Parity error during issue?
        BNE.S   @30
        TST.B   CMD_BUFFER(A3)          ;Write command?
        BNE.S   @10                     ;Skip if yes
        MOVE.W  #RD_NEXT-STATE_TABLE,STATE(A3)  ;Continue with read state next
        MOVE.B  #$22,ER_HS(A3)          ;Expected response
        ORI.B   #$18,ORB(A4)            ;reset dir=in ; cmd=false
        CLR.B   DDRA(A4)                ;set port A bits to input
        BRA.S   @20                     ;Go finish state

@10     MOVE.W  #WRT_NEXT-STATE_TABLE,STATE(A3) ;Go to write state next
        MOVE.B  #$23,ER_HS(A3)      ;Expected response

@20    ;finish off sending request
```

```
        MOVEQ   #1,D0                   ;Advance to next state now
        RTS

@30     MOVE    #PRTY_ERR,ERROR(A3)
        ORI.B   #$18,ORB(A4)            ;reset dir=in ; cmd=false
        CLR.B   DDRA(A4)                ;set port A bits to input
        MOVEQ   #0,D0                   ;RETURN NOW
        RTS

;S50NONIO ;A non-I/O request is desired
;       MOVE    #FMTCMD-STATE_TABLE,STATE(A3)
;       MOVEQ   #1,D0                   ;Next state now
;       RTS


        ;---------------------------------------------------------
        ;READ DATA - FROM MULTI-BLOCK COMMAND
        ;  return: continue in HS state, or return without error or with parity
        ;          or checksum error

S51     MOVEQ   #0,D1                   ;INIT CHECKSUM VALUE
        BSR     RDDATA                  ;READ DATA FIRST
        BSR     RDHDR                   ;THEN HEADER
        TST.B   D1                      ;NOW SEE IF CHECKSUM IS ZERO
        BNE     S51CS_BAD

S51CS_OK

;       BTST    #3,IFR(A4)
;       BNE     S51PE                   ;PARITY ERROR?

        CMPI.B  #$09,STATUS(A3)
        BEQ     S51CRC                  ;RETURN WITH CHECKSUM ERROR IF GOT CRC ERROR

        ;Continue with next block in current request (cxfercnt >= 0)
        ;advance to next state (cxfercnt = 0)

        SUBQ.L  #1,SECT_LEFT(A3)        ;decr sector transfer count
        BSR     ADJ_HDR                 ;call routine for header adjustment
        BSR     ADJ_NEXTBLK             ;and setup for next block
        ADDQ.L  #1,SECTOR(A3)           ;update logical block ptr
        MOVEQ   #1,D0                   ;Advance to next state now
        SUBQ.B  #1,CXFERCNT(A3)         ;Decrement current transfer count
        BEQ.S   @30                     ;Skip if no more blocks in current request
        TST.L   STATUS(A3)              ;Any error at all on last transfer?
        BNE.S   @31                     ;Skip if yes
        MOVE.W  #RD_NEXT-STATE_TABLE,STATE(A3) ;Read next block
        MOVE.B  #$22,ER_HS(A3)          ;Expected response
        RTS

@30     ;Current command completed
        ADDQ    #2,STATE(A3)            ;Next state to free device
        MOVE.B  #1,ER_HS(A3)
        RTS

@31     ;Current command stopped by firmware due to non-fatal error (reissue cmd)
```

```
        MOVE    #MULTI_CMD-STATE_TABLE,STATE(A3) ; next state
        RTS

        ;Handle various read errors
S51PE   ;Parity error
        MOVE    #PRTY_ERR,ERROR(A3)
        BRA     S51QUIT

S51CRC  ;CRC error
        MOVE    #CS_ERR,ERROR(A3)
S51QUIT MOVEQ   #0,D0
        RTS

        ;These errors require the firmware to be stopped
S51CS_BAD  ;Software checksum is bad
        MOVE    #CS_ERR,ERROR(A3)

S51HDR_BAD ;Header error - error code is already in ERROR
        MOVE.B  #0,CXFERCNT(A2)         ;Clear current transfer count
        MOVE.L  D4,A0
        MOVE.L  #0,SECT_LEFT(A3)        ;Clear sectors left to transfer
        ADDQ    #2,STATE(A3)            ;Next state
        MOVEQ   #1,D0                   ;NOW
        MOVE.B  #$FF,ER_HS(A3)          ;Any response is OK
        RTS

        ;---------------------------------------------------------------
        ;ISSUE NEW MULTI-BLOCK COMMAND IF NECESSARY
        ;  return: sect_left = 0 - return immediately
        ;          sect_left<> 0 - next state = NEW_CMD immediately

S52     TST.L   SECT_LEFT(A3)           ;Sectors still to transfer?
        BEQ.S   @1                      ;Skip if no
        MOVE    #MULTI_CMD-STATE_TABLE,STATE(A3) ; next state
        MOVEQ   #1,D0                   ;Do it now
        RTS
@1      MOVEQ   #0,D0                   ;Return immediately
        RTS

        ;---------------------------------------------------------------
        ;CHECK RESPONSE DURING MULTI-BLOCK WRITE
        ;  return: cxfercnt < 0 - advance to next state after interrupt
        ;          cxfercnt >=0 - next state = WRT_NEXT after interrupt
        ;          response = $A3 - next state = WRT_STATUS after interrupt
        ;          response <>$A3 - next state = BDR immediately

S53     MOVE.B  PORTA(A4),D1            ;Read response
        ANDI.B  #$E7,ORB(A4)            ;Dir=out; cmd=true
        MOVE.B  #$FF,DDRA(A4)           ;Port A bits to output
        MOVE.B  #$55,PORTA(A4)          ;Send reply

;        ORI.B   #$01,PCR(A4)            ;Interrupt on rising edge

        MOVE.B  #$02,IFR(A4)            ;Clear interrupt flag
        ORI.B   #$10,ORB(A4)            ;Cmd = false
```

```
        ;Determine next state
        CMPI.B  #$23,D1                 ;Expected response?
        BNE.S   @20                     ;Skip if no

        BSR     ADJ_HDR                 ;adjust header ptr for next block
        TST.B   INITRQST(A6)            ;doing initialize?
        BEQ.S   @4                      ;skip if not
        MOVE.L  A0,-(SP)                ;save reset ptr
        MOVE.L  D4,A0                   ;get param blk ptr
        MOVE.L  IOBUFFER(A0),A2         ;else restore buffer ptr
        MOVE.L  (SP)+,A0                ;restore reset ptr
        BRA.S   @5                      ;and continue

@4      BSR.S   ADJ_NEXTBLK             ;and setup for next block

@5      ADDQ.L  #1,SECTOR(A3)           ;update logical block ptr for next command
        SUBQ.B  #1,CXFERCNT(A3)         ;Decrement current transfer count
        BEQ.S   @10                     ;Skip if no more sectors
        MOVE.W  #WRT_NEXT-STATE_TABLE,STATE(A3) ;next state

;-------------------------------------------------------------------------------
@6      BSR     WAIT_NOTBUSY            ;poll until not busy
        CLR.B   DDRA(A4)                ; set port A bits to input
        ORI.B   #$18,ORB(A4)            ; set dir = in, cmd = false
        TST     D0
        BEQ.S   @7                      ;skip if OK
        MOVE.B  #TIME_ERR,ERROR(A3)     ;else set timeout error
        MOVEQ   #0,D0                   ;return now
        RTS                             ;and exit
@7      MOVEQ   #1,D0                   ; continue at next state
        RTS
;-------------------------------------------------------------------------------
;@6      MOVE    #WAIT_INT,ERROR(A3)     ; wait for interrupt
;       MOVE.B  #$FF,T2CH(A4)           ;START TIMER FOR DISCON ERROR CHECK
;       MOVEQ   #0,D0
;       RTS
;-------------------------------------------------------------------------------

@10     ;no more blocks in current request
        MOVE.B  #$27,ER_HS(A3)          ;Expected response
        ADDQ    #2,STATE(A3)            ;advance to next state
        BRA.S   @6                      ;wait for interrupt

@20     ;not expected response
        CMPI.B  #$A3,D1                 ;Error response?
        BNE.S   @30                     ;skip if no
        MOVE.W  #WRT_STATUS-STATE_TABLE,STATE(A3) ; next state
        BRA.S   @6                      ;wait for interrupt

@30     ;bad response
        MOVE.W  #BDR-STATE_TABLE,STATE(A3) ;next state
        MOVEQ   #1,D0                   ;do it now
        RTS

;-------------------------------------------------------------------------
; Entry point for controller initialization
```

```
;  Makes no assumptions about state of controller
;  Assumes A3 = ptr to drive locals
;          A4 = base address of port's VIA
;          A0 = base address for reset/parity reset lines
;  Returns D0 = result

S80     CLR.L   D3
        MOVE.B  HWDDRB(A3),D3   ;get offset for reset line direction reg
        ORI.B   #$A0,0(A0,D3)   ;SET PROFILE-RESET & PARITY-RESET TO OUTPUT
        ORI.B   #$A0,ORB(A0)    ;NORMALLY BITS 5 & 7 OF ORB = 1
        ANDI.B  #$7B,PCR(A4)
        ORI.B   #$6B,PCR(A4)    ;SET PCR TO $6B, WITHOUT CHANGING BIT #4
        MOVEQ   #0,D0           ;PUT 0 IN D0
        CLR.B   DDRA(A4)
        ANDI.B  #$FC,DDRB(A4)
        ORI.B   #$1C,DDRB(A4)
        ANDI.B  #$FB,ORB(A4)
        ORI.B   #$18,ORB(A4)
        BTST    #0,IRB(A4)      ;DISCONNECTED?
        BEQ.S   @2
        MOVE    #NODISK_ERR,D0  ;DISCONNECTED ERROR
        BRA     BYE             ;RETURN

        ;TRY READ TO MAKE SURE IT IS A PROFILE AND GET DEVICE CHARACTERISTICS!

@2      BSR     WAIT_NB1                ;Wait for device not busy
        TST     D0                      ;Timeout?
        BEQ.S   DOIT
        MOVE    #TIME_ERR,D0            ;Timeout error - give up
        BRA     BSYXT

DOIT    BSR     STRTRD                  ;begin read operation
        BEQ.S   GOTIT                   ;skip if successful
        BSR     WAIT_NOTBUSY            ;else wait for not busy
        BSR     STRTRD                  ;and try again
        BEQ.S   GOTIT                   ;skip if OK
        BSR     DORESET                 ;else try doing reset
        BSR     WAIT_NOTBUSY            ;wait until ready
        BSR     STRTRD                  ;and try again
        BEQ.S   GOTIT                   ;skip if successful
        TST.W   STATUS+2(A3)            ;check if reset error
        BPL.S   BSYXT                   ;error if not
        BSR.S   WAIT_NOTBUSY            ;else wait for not busy
        BSR     STRTRD                  ;and try final time
        BNE.S   BSYXT                   ;give up if still no good

GOTIT   MOVEQ   #13,D0                  ;Read 14 bytes
@2      MOVE.B  IRA(A4),D1
        DBF     D0,@2
        MOVE.B  IRA(A4),DRIVETYPE(A3)   ;Read next byte as drivetype
        MOVEQ   #2,D0                   ;Read 3 bytes
@3      MOVE.B  IRA(A4),D1
        DBF     D0,@3

        ;Save drive size in drive queue element
```

```
        MOVEQ   #DQEL,D2                ;set offset ptr to drive queue element
        MOVE.B  IRA(A4),D0              ;ignore first byte
        MOVE.B  IRA(A4),DQDRVSIZE(A3,D2)  ;save last 2 bytes of size
        MOVE.B  IRA(A4),DQDRVSIZE+1(A3,D2)
        MOVE.W  DQDRVSIZE(A3,D2),REALSIZE(A3)  ;save copy also

        ;Set as non-ejectable disk

        MOVEQ   #StopEject,D0
        OR.W    D0,INFO1(A3)            ;set indicator

        ;Finally add drive to drive queue if not already there

        MOVE.W  DRIVE(A6),D1            ;get drive #
        MOVE.L  DrvQHdr+QHead,D0        ; find the drive queue
@4      BEQ.S   @6                      ; skip if no more entries
        MOVE.L  D0,A0                   ; else get entry ptr
        CMP.W   DQDrive(A0),D1          ; drive installed in this entry?
        BEQ.S   @7                      ; skip if yes
@5      MOVE.L  QLink(A0),D0            ; else get next entry ptr
        BRA.S   @4

@6      MOVE.W  D1,D0                   ;get drive #
        SWAP    D0
        MOVE.W  DCtlRefNum(A1),D0       ;add driver refnum
        LEA     DQEL(A3),A0             ;set ptr to drive queue element
        _AddDrive                       ;go put into queue

@7      MOVEQ   #0,D0                   ;Error code = 0

BSYXT   CLR.B   DDRA(A4)                ; Set Port A bits to input
        ORI.B   #$18,ORB(A4)            ; set dir=in, cmd=false

BYE     MOVE    D0,ERROR(A3)            ;save result
        MOVEQ   #0,D0                   ;return now
        RTS


        ;---------------------------------------------------------------
        ;Poll until device not busy or timeout
        ;Registers used = D0, D1
        ;Output - D0 = 0 OK to continue
        ;           = 1 Timeout

WAIT_NOTBUSY
        MOVE.L  #RDTIME,D1              ;normal wait time (about 8 secs)
        BRA.S   WAITCHK
WAIT_NB1
        MOVE.L  #STRTTIME,D1            ;power-up time (about 3 mins max)
WAITCHK
        MOVEQ   #0,D0                   ;OK to continue
@1      BTST    #1,IRB(A4)              ;Busy?
        BNE.S   @2
        SUBQ.L  #1,D1
        BNE.S   @1
        MOVEQ   #1,D0                   ;TIMEOUT
```

```
@2      RTS

        ;-------------------------------------------------------------
        ;POLL UNTIL DEVICE IS BUSY FOR 32 CHECKS IN A ROW
        ;Registers used = D0, D1
        ;Output - D0 = 0 OK to continue
        ;           = 1 Timeout

WAIT_BUSY
        MOVEQ   #0,D0                   ;OK to continue
        MOVE.L  #RDTIME,D1
@1      BTST    #1,IRB(A4)              ;Not Busy?
        BEQ.S   @2
        SUBQ.L  #1,D1
        BNE.S   @1
        MOVEQ   #1,D0                   ;TIMEOUT
@2      RTS

        ;-------------------------------------------------------------
        ; Do read operation
        ; Registers used = D0, D2

STRTRD  MOVEQ   #1,D2                   ;Expected response
        BSR.S   DOSHAKE                 ;Do hand shake
        BNE.S   STRTXIT                 ;skip if error

        ;Get Device Characteristics

        MOVEQ   #0,D0                   ;assume no error
        MOVE.B  D0,ORA(A4)              ;Command=Read
        NOP
        MOVE.B  #$FF,ORA(A4)            ;Read sector = $FFFFFF
        MOVE.B  #$FF,ORA(A4)
        MOVE.B  #$FF,ORA(A4)
        MOVE.B  #$0A,ORA(A4)            ;Retry count
        MOVE.B  D0,ORA(A4)              ;Sparing threshold

GTTYP   MOVEQ   #2,D2                   ;Expected response
        BSR     DOSHAKE                 ;Do hand shake
        BNE.S   STRTXIT                 ;skip if error
        CLR.B   DDRA(A4)                ;Set Port A bits to input
        ORI.B   #$18,ORB(A4)            ;set dir=in; cmd=false

        MOVE.B  IRA(A4),STATUS(A3)      ;Read 4 bytes of status
        MOVE.B  IRA(A4),STATUS+1(A3)
        MOVE.B  IRA(A4),STATUS+2(A3)
        MOVE.B  IRA(A4),STATUS+3(A3)

        MOVE.L  #$C140C000,D0           ;check for fatal error
        AND.L   STATUS(A3),D0           ;setup exit result
        BEQ.S   STRTXIT                 ;exit if OK
        MOVE    #HD_ERR,D0              ;else set error code
STRTXIT RTS                             ;and return

        ;-------------------------------------------------
        ; HandShake routine
```

```
        ; Input - D2 = Expected response
        ;       - A4 = Hardware Base Address
        ; Output- D0 = error # (=0 if OK)
        ; Exits with dir=out, Port A to output, cmd=fals

DOSHAKE ANDI.B  #$EF,ORB(A4)    ;Cmd=true
        ORI.B   #$08,ORB(A4)    ;Set dir = in
        CLR.B   DDRA(A4)        ;Set Port A bits to input
        BSR     WAIT_BUSY       ;Wait for device to go busy
        TST     D0              ;Timeout?
        BEQ.S   GTRSP           ;Skip if OK
        MOVE    #TIME_ERR,D0    ;Timeout error
        RTS                     ;Exit early

GTRSP   MOVE.B  PORTA(A4),D1    ;Get response
        ANDI.B  #$E7,ORB(A4)    ;Dir=out; cmd=true
        MOVE.B  #$FF,DDRA(A4)   ;Port A to output
        CMP.B   D2,D1           ;Compare response
        BNE.S   RS_BAD          ;Skip if bad response
        MOVE.B  #$55,PORTA(A4)  ;Send OK reply
        ORI.B   #$10,ORB(A4)    ;Cmd=false
        BSR     WAIT_NOTBUSY    ;Wait for not busy
        TST     D0              ;Timeout?
        BEQ.S   @1
        MOVE    #TIME_ERR,D0    ;timeout error
@1      RTS

RS_BAD  ;bad response
        MOVE.B  #$AA,PORTA(A4)  ;Send negative reply
        ORI.B   #$10,ORB(A4)    ;Cmd=false
        BSR     WAIT_NOTBUSY    ;Wait for not busy
        MOVE    #RESP_ERR,D0      ;Bad response error code
        RTS

        ;-------------------------------------------------
        ;  Routine to do controller reset
        ;  Assumes A0 = base address for reset port
        ;
DORESET ANDI.B  #$7F,ORB(A0)    ;set reset signal
        MOVEQ   #$7F,D0         ;delay for about .1 sec
@1      SUBQ    #1,D0
        BNE.S   @1
        ORI.B   #$80,ORB(A0)    ;remove reset signal
        RTS                     ;and exit

        .END
```

```
; File: HDMount.Text
;_____________________________________________________________
;
; Hard Disk Driver Mount Routine
;
;    written by Rich Castro
;
;  This program attempts to mount a disk.
;
;  Modification History:
;   3 May 84  RDC  Write initial version
;  16 May 84  RDC  Add disk size check before trying mount
;  13 Nov 84  KWK  Check for volume already mounted error
;_____________________________________________________________

          .NOLIST
          .INCLUDE TLASM-SYSEQU.TEXT
          .INCLUDE TLASM-SYSMACS.TEXT
          .INCLUDE TLASM-SYSERR.TEXT
          .INCLUDE TLASM-TOOLMACS.TEXT
          .INCLUDE TLASM-TOOLEQU.TEXT
          .LIST


NoDrvErr  .EQU     4                ;no drive found
NoSpace   .EQU     6                ;no space available on drive

MinSize   .EQU     400              ;minimum # of blocks for creating Mac disk

          .FUNC    HDMount,0

;  on entry 4(SP)=result
;  now try to mount hard disk if one was found

;  first search drive queue for default drive

          MOVEQ    #HDDrive,D1      ; drive we're looking for
          MOVE.L   DrvQHdr+QHead,D0 ; find the drive queue element
@2        BEQ      NoDrvFnd         ; exit if drive not found in drive queue (last QLink = 0)
          MOVE.L   D0,A1
          CMP.W    DQDrive(A1),D1   ; this entry?
          BEQ.S    @4               ; br if so
          MOVE.L   QLink(A1),D0
          BRA.S    @2

@4        CMPI.W   #MinSize,DQDrvSize(A1)  ;enough space available on disk?
          BLT.S    SpaceErr                ;exit if not

;  found it - create parameter block and try to mount it

@3        MOVEQ    #(IOVQElSize/2)-1,D0
@1        CLR.W    -(SP)            ; clear a parameter block on the stack
          DBRA     D0,@1

          MOVE.L   SP,A0                   ; set ptr to parm block
          MOVE.L   DQDrive(A1),IODrvNum(A0) ; fill in IODrvNum, IORefNum
```

```
          _MountVol
          ADD       #IOVQElSize,SP      ; restore stack
          CMP.W     #VolOnLinErr,D0     ; already mounted?
          BNE.S     HDMXIT              ; no, return w/result
          MOVEQ     #0,D0               ; no error
          BRA.S     HDMXIT              ; and exit

;  exit with results

NoDrvFnd  MOVEQ     #NODrvErr,D0        ;set no drive error
          BRA.S     HDMXIT

SpaceErr  MOVEQ     #NoSpace,D0         ;set space error

HDMXIT    MOVE.L    (SP)+,A0            ;get ret addr
          MOVE      D0,(SP)             ;save result
          JMP       (A0)                ;and return

          .END
```

```
; File MWInstall-MWI/MountTable.text
;
;    Monitor mount table/volume entry for MacWorks profile
;
;    Modification History
;
;   29-Oct-84  New Today
;
;

     .LONG     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0   ; 0..$37  Reserved
     .BYTE     7                                 ; $38..$3F (length of name)
     .ASCII    'KRUGLER'                         ; Name of disk

; Mount table.  Each entry is 4 bytes long and corresponds to
; the unit entries 0..20.  Bits 31..29 = drive number, 28..16
; = first block #/8, 15 = write protect bit, 14..13 = 0's,
; 12..0 = one past last block #/8.
; The MacWorks image is unit #5

     .LONG     0,0,0,0,0                         ; Unit #'s 0..4
     .WORD     $2001                             ; Drive 1, start block = 8
     .WORD     $0065                             ; No WrtProt, last blk+1 = 808
     .LONG     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0   ; Unit #'s 6..20

     .BLOCK    108,0                             ; 108 0's to fill to $FF

; Volume entries.  Each entry is the same as a mount table entry.
; The MacWorks volume is the first one

     .WORD     $2001                             ; Drive 1, start block = 8
     .WORD     $0065                             ; No WrtProt, last blk+1 = 808

; The marker for no more volume entries has drive # = 7 and all else = 0

     .WORD     $E000                             ; Drive 7, rest = 0
     .WORD     $0000                             ;

     .BLOCK    248,0                             ; fill rest of 256 bytes with
                                                 ; all zeros
```

```
; File: HDskOpen.Text
;_____________________________________________________________
;
; Hard Disk Driver Boot Routine
;
;    written by Rich Castro  3/18/84
;
;  This program is executed at system init time on a Lisa system
;  running MacWorks.  It uses the HDOpen routine which checks for hard
;  disks attached to the Lisa and mounts them for use.  If any errors
;  occur, the program aborts with the error code in D0.
;
;  Modification History:
;   3 May 84  RDC  Add support for Lisa/Mac disk sharing
;  15 May 84  RDC  Add calls to deallocate INIT routine when done
;  11 Jun 84  RDC  Add code to switch to hard disk as new boot device
;  21 Jun 84  RDC  Change disk driver id to 1
;  21 Jun 84  RDC  Add code to check for driver already installed
;  21 Jun 84  RDC  Change driver name to .HardDisk
;   2 Jul 84  RDC  Skip controller init if driver already installed
;  26 Oct 84  KWK  Equate cleanup
;_____________________________________________________________

          .NOLIST
          .INCLUDE TLASM-SYSEQU.TEXT
          .INCLUDE TLASM-SYSMACS.TEXT
          .INCLUDE TLASM-SYSERR.TEXT
          .INCLUDE TLASM-TOOLMACS.TEXT
          .INCLUDE TLASM-TOOLEQU.TEXT
          .LIST

SYSID     .EQU      $400009             ;system ID location
LisaID    .EQU      $FF                 ;ID for Lisa running MacWorks
NotLisaEr .EQU      1                   ;not Lisa error
OpenDErr  .EQU      2                   ;open driver error
ResrcErr  .EQU      3                   ;get resource error
NoDrvErr  .EQU      4                   ;no drive found
InitDErr  .EQU      5                   ;init controller error

AllResfiles .EQU    0                   ;resource mgr code for all res files


          .PROC     HDskOpen,0

;  first test to make sure we're running on a Lisa system

          MOVEM.L   D2-D7/A2-A6,-(SP)   ;preserve registers
          MOVE.B    SYSID,D0            ;read system id
          CMP.B     #LisaID,D0          ;is it a Lisa?
          BNE       NotLisa             ;exit if not

;  we're on the right system - allocate space for parameter block

          LEA       PARAMBLK,A0         ;ptr to save area
          MOVEQ     #(IOFQElSize/2)-1,D0
@1        CLR.W     -(SP)               ; clear a parameter block on the stack
```

```
        DBRA    D0,@1
        MOVE.L  SP,(A0)             ; save for IO calls

;  next check to see if driver already opened (by new MacWorks)

        MOVE    #HDskID,D2          ;driver id
        ASL.W   #2,D2               ; multiply by four
        MOVE.L  UTableBase,A1       ; get address of unit table
        MOVE.L  0(A1,D2.W),D2       ; add in the offset
        BNE     HDOXIT              ; exit if driver already installed

;  driver not installed - let's try to open it

@10     LEA     HDDName,A1          ;setup ptr to name
        MOVE.L  A1,IOFileName(SP)   ;save in param block
        CLR.B   IOPermssn(SP)       ; r/w permissions
        MOVE.L  SP,A0
        _Open                       ; open it!
        BNE     OpenFail            ; exit if open error

;  Open worked - do detach resource to ensure we don't get purged

        SUBQ    #4,SP               ;make room for result
        MOVE.L  #'DRVR',-(SP)       ;setup type
        MOVE    #HDskID,-(SP)       ; and id
        _GetResource

        MOVE.L  (SP),A2             ;save handle
        MOVE.L  A2,D0               ;set condition code
        BEQ     ResrcFail           ;exit if didn't get it
        _DetachResource

;  all OK so far - now init the drive's controller and add to drive queue if found

@20     MOVE.L  ParamBlk,A0           ;get ptr to param block
        MOVE.W  #HDDrive,IODrvNum(A0) ;set for builtin drive
        MOVE.W  #InitCode,CsCode(A0)  ;request controller init operation
        _Control                      ;do as Control call to driver
        TST     D0                    ;check result
        BEQ.S   @4                    ;skip if OK
        CMP.W   #NODriveErr,D0        ;no drive found?
        BEQ     NoDrvFnd              ;exit if yes
        CMP.W   #LisaDskErr,D0        ;Lisa disk found?
        BEQ.S   @4                    ;OK - try to mount to see if shared disk
        BRA     InitFail              ;else go to general error

;  now try to mount hard disk if one was found

@4      MOVEQ   #HDDrive,D1           ; drive we're looking for
        MOVEQ   #NSDrvErr,D3        ; assume we can't find it in the drive q

;  search drive queue for default drive (HDDrive)

        MOVE.L  DrvQHdr+QHead,D0    ; find the drive queue element
@2      BEQ     NoDrvFnd            ; exit if drive not found in drive queue (last QLink = 0)
        MOVE.L  D0,A1
```

```
        CMP.W   DQDrive(A1),D1      ; this entry?
        BEQ.S   @3                  ; br if so
        MOVE.L  QLink(A1),D0
        BRA.S   @2

;  found it - let's try to mount it

@3      MOVE.L  DQDrive(A1),IODrvNum(SP) ; fill in IODrvNum, IORefNum
        MOVE.L  ParamBlk,A0         ; set ptr to parm block
        _MountVol
        BNE.S   HD0XIT              ; exit if error

;  mounted, now let's try to switch over to the system file
;    First, make sure the hard disk has SYSTEM and FINDER files

        lea     Systemname,a1
        move.l  a1,ioFileName(a0)
        clr.w   ioFDirIndex(a0)
        clr.w   ioFileType(a0)
        _GetFileInfo
        bne.s   HD0xit

        lea     Fndrname,a1
        move.l  a1,ioFileName(a0)
        clr.w   ioFDirIndex(a0)
        clr.w   ioFileType(a0)
        _GetFileInfo
        bne.s   HD0xit

;    both SYSTEM and FINDER are on the disk, so now we can switch over to them

        MOVE.L  ParamBlk,A0         ; set ptr to parm block
        CLR.L   IOVNPtr(A0)         ; don't use volume name!
        _SetVol                     ; Set default volume=system.

        MOVE.W  #AllResFiles, -(SP) ; Push allResFile ID
        _CloseResFile               ; and close them all!
        CLR.L   ResErrProc          ; Don't bother with errors

        SUBQ    #2, SP              ; Save space for integer result
        _InitResources              ; Init the resource manager
        ADDQ    #2, SP              ; Ignore result code
        _InitFonts                  ; Initialize the font mgr

        MOVE.W  #HDDrive,BootDrive  ; Reset the bootDrive global
        BRA.S   HD0xit

;  exit with results

NotLisa MOVEQ   #NotLisaEr,D0       ;set error code
        BRA.S   HD1XIT

OpenFail MOVEQ  #OpenDErr,D0        ;set error code
        BRA.S   HD0XIT

InitFail MOVEQ  #InitDErr,D0        ;set error code
```

```
          BRA.S     HDOXIT

ResrcFail MOVEQ     #ResrcErr,D0        ;set error code
          BRA.S     HDOXIT

NoDrvFnd  MOVEQ     #NODrvErr,D0        ;set error code

HDOXIT    ADD       #IOFQElSize,SP      ;restore stack

;  release the program space before exiting

HD1XIT    MOVE.L    D0,-(SP)            ;save return code
          LEA       HDskOpen,A0         ;get ptr to start of code
          _RecoverHandle  ,SYS          ;get handle
          _HUnlock                      ;tell memory manager to unlock
           BNE.S     @1                 ;skip if error
          _HPurge                       ;and make routine purgable

@1        MOVE.L    (SP)+,D0            ;restore return code
          MOVEM.L   (SP)+,D2-D7/A2-A6   ;restore other regs
          RTS                           ;and return

; data area

PARAMBLK  .LONG     0                   ;ptr to param block
HDDName   .BYTE     9                   ;# of bytes in name
          .Ascii    '.HardDisk'         ;filename of driver
SystemName .BYTE    6
          .Ascii    'SYSTEM'
Fndrname  .BYTE     6
          .Ascii    'FINDER'

          .END
```

```
; File: HDOpen.Text
;_____________________________________________________________________
;
; Hard Disk Driver Open/Mount Routine (also check if on hard disk)
;
;     written by Rich Castro
;
;  This program opens the hard disk driver for use on a Lisa system
;  running MacWorks.  Following a successful open, the hard disk is
;  also mounted for use.  Errors are returned for the following cases:
;
;    1) Not a Lisa system
;    2) Open driver failure
;    3) Mount drive failure
;    4) No disk found
;
;  Modification History:
;   8 Mar 84  RDC  Write initial version
;  18 Mar 84  RDC  Add no disk found error
;  26 Mar 84  RDC  Add check to see if driver already open
;   9 Apr 84  RDC  Add Control call for init device
;  10 Apr 84  RDC  Add fetch of driver refnum if already open
;  10 May 84  RDC  Allow init error to return actual error code from driver
;  14 May 84  RDC  Change not Lisa error code (conflicts with UserCancel code)
;  21 Jun 84  RDC  Change disk driver id to 1
;  21 Jun 84  RDC  Change driver name to .HardDisk
;  18 Jul 84  RDC  Add preliminary check to see if drive already mounted
;  26 Oct 84  KWK  Constants cleanup
;   9 Nov 84  KWK  Stripped all code dealing w/driver installation & drive already mounted stuff
;  12 Dec 84  KWK  Moved Lisa equates to tlasm-sysequ.text
;  (18Apr85)  RDC  Change MacWorks revision check to allow rev C or later (>= $80FF)
;
;  To add: return indication to caller of drives successfully mounted
;_____________________________________________________________________

          .NOLIST
          .INCLUDE TLASM-SYSEQU.TEXT
          .INCLUDE TLASM-FSEQU.TEXT
          .INCLUDE TLASM-SYSMACS.TEXT
          .INCLUDE TLASM-SYSERR.TEXT
          .INCLUDE TLASM-TOOLMACS.TEXT
          .INCLUDE TLASM-TOOLEQU.TEXT
          .LIST

OpenDErr  .EQU     2                ;open driver error
ResrcErr  .EQU     3                ;get resource error
NoDrvErr  .EQU     4                ;no drive found
NtLisaErr .EQU     5                ;not Lisa error
NtRevCErr .EQU     7                ;not Rev C error
OnHDiskEr .EQU     8                ;running on a hard disk error


          .FUNC    HDOpen,0

;  on entry 4(SP)=result
;  first test to make sure we're running on a Lisa system
```

```
        MOVEM.L   D2-D7/A1-A6,-(SP)   ;preserve registers
        MOVE.W    SYSID,D0            ;read system id
        CMP.B     #LisaID,D0          ;is it a Lisa?
        BNE       NotLisa             ;exit if not

;  on a Lisa, check version number of macworks

        CMP.W     #RevCID,D0          ;is it a RevC or later MacWorks?          (18Apr85)
        BLO       NotRevC             ;exit if not                              (18Apr85)

;  now check if we're running on a hard disk

        MOVE.W    CurApRefnum,D0      ; get the application file refnum
        MOVE.L    FCBSPtr,A0          ; get ptr to file control block start
        MOVE.L    FCBVPtr(A0,D0.W),A0 ; get ptr to volume control block
        MOVE.W    VCBDrefnum(A0),D0   ; get volume driver refnum
        CMP.W     #HDRfNum,D0         ; is it the hard disk driver?
        BEQ       OnHDisk             ; yes, exit w/error

;  everything is cool so far - allocate space for parameter block

        LEA       PARAMBLK,A0         ;ptr to save area
        MOVEQ     #(IOVQElSize/2)-1,D0
@1      CLR.W     -(SP)               ; clear a parameter block on the stack
        DBRA      D0,@1
        MOVE.L    SP,(A0)             ; save for IO calls

;  driver has been installed by Rev C boot disk & disk unmounted before we get here
;  Set up the DCE handle

        MOVE.W    #HDskID,D2          ; driver ID
        ASL.W     #2,D2               ; get offset into unit table
        MOVE.L    UTableBase,A1       ; get ptr to unit table
        MOVE.L    0(A1,D2.W),D2       ; get handle

;  drive not mounted - do setup for device initialization

@12     MOVE.L    D2,A1                       ; get DCE handle
        MOVE.L    (A1),A1                     ; and dereference
        MOVE.L    ParamBlk,A0                 ; get ptr to param block
        MOVE.W    DCtlRefNum(A1),IORefNum(A0) ; get driver refnum

;  now init the drive's controller and add to drive queue if found

@20     MOVE.L    ParamBlk,A0           ;get ptr to param block
        MOVE.W    #InitCode,CsCode(A0)  ;request controller init operation
        MOVE.W    #HDDrive,IODrvNum(A0) ;only for builtin drive for now
        _Control                        ;do as Control call to driver

        TST       D0                    ;check result
        BEQ.S     @4                    ;skip if OK
        CMP.W     #NODriveErr,D0        ;no drive found?
        BEQ.S     NoDrvFnd              ;exit if yes
        BRA.S     HDOXIT                ;else go to general error
```

```
;  now try to mount hard disk if one was found

@4      MOVEQ   #HDDrive,D1         ; drive we're looking for
        MOVEQ   #NSDrvErr,D3        ; assume we can't find it in the drive q

;  search drive queue for default drive (HDDrive)

        MOVE.L  DrvQHdr+QHead,D0    ; find the drive queue element
@2      BEQ     NoDrvFnd            ; exit if drive not found in drive queue (last QLink = 0)
        MOVE.L  D0,A1
        CMP.W   DQDrive(A1),D1      ; this entry?
        BEQ.S   @3                  ; br if so
        MOVE.L  QLink(A1),D0
        BRA.S   @2

;  found it - let's try to mount it

@3      MOVE.L  DQDrive(A1),IODrvNum(SP) ; fill in IODrvNum, IORefNum

        MOVE.L  ParamBlk,A0         ; set ptr to parm block
        _MountVol
        BRA.S   HD0XIT              ; exit with result in D0

;  exit with results

NotLisa MOVEQ   #NtLisaErr,D0       ;set error code
        BRA.S   HD1XIT

NotRevC MOVEQ   #NtRevCErr,D0       ;set error code
        BRA.S   HD1XIT

OnHDisk MOVEQ   #OnHDiskErr,D0      ;set error code
        BRA.S   HD1XIT

MntErr  MOVEQ   #VolOnLinErr,D0     ;set error code
        BRA.S   HD0XIT

OpenFail MOVEQ  #OpenDErr,D0        ;set error code
        BRA.S   HD0XIT

ResrcFail MOVEQ #ResrcErr,D0        ;set error code
        BRA.S   HD0XIT

NoDrvFnd MOVEQ  #NODrvErr,D0        ;set error code

HD0XIT  ADD     #IOVQElSize,SP      ;restore stack

HD1XIT  MOVEM.L (SP)+,D2-D7/A1-A6   ;restore regs
        MOVE.L  (SP)+,A0            ;get ret addr
        MOVE    D0,(SP)             ;save result
        JMP     (A0)                ;and return

; data area

PARAMBLK .LONG  0                   ;ptr to param block
HDDName .BYTE   9                   ;# of bytes in name
```

```
.Ascii    '.HardDisk'          ;filename of driver

.END
```

```
; File: HDFormat.Text
;_____________________________________________________________________
;
; Hard Disk Format Routine
;
;     written by Rich Castro
;     adapted from Disk Initialization Package by Larry Kenyon
;
;  This program formats the hard disk.
;
;  Assumption made that hard disk driver installed before this program
;  is called.
;
;  Stack usage as follows (as offset from stack ptr after LINK):
;
;        10      Function result
;        8       Disk type
;        4       Return address
;        0       Old A6
;       -4       Stack Buffer ptr
;       -8       Parameter block ptr
;       -10      Save of new mount error
;       -12      Save of drive # (Unused???)
;       -14      Disk type (Unused???)
;       -16      Drive queue element ptr
;       -18      Dialog result code
;       -22      Save of caller's grafport
;       -534     Buffer area for writing directory
;       -598     Parameter block area
;
;
; Modification History:
;  7 Mar 84  RDC  Write initial version
;  9 Mar 84  RDC  Move driver open/mount code to HDOpen routine
; 12 Mar 84  RDC  Make into a Pascal callable routine
; 20 Mar 84  RDC  Change directory initialization parameters
; 26 Mar 84  RDC  Remove _Eject calls which don't make sense for hard disk, add
;                 change of cursor displayed as needed
;  7 Apr 84  RDC  Add fix for bus error caused by InitCursor call
;  9 Apr 84  RDC  Change allocation block size to 4K (8 blocks)
; 11 May 84  RDC  Add support for reinitialize of Mac disk
; 15 May 84  RDC  Add support for reinitialize of Lisa disk (erase cmd)
;            RDC  Increase max file directory size to 148 blocks (1036 files)
; 16 May 84  RDC  Add drive type parameter
; 24 May 84  RDC  Add fix for directory calculations
; 29 May 84  RDC  Add second fix for write directory routine to allow 1 block MDB
;  6 Jun 84  RDC  Delete initialize dialogs; moved to HDUtil for easier manipulation
;  7 Jun 84  RDC  Add time calculation for format operation
; 18 Jun 84  RDC  Add set of watch cursor while directory being written
;  9 Nov 84  KWK  Added setup of vol/driver refnums before UnMountVol/Control calls
; 17 Nov 84  KWK  Moved disk naming/directory writing stuff to HD/Name
; 17 Nov 84  KWK  Removed error/disk type parameters, cleaned up equates
; 18 Nov 84  KWK  Partial replace of disk type for physical format of entire disk.
;_____________________________________________________________________

        .NOLIST
```

```
        .INCLUDE TLASM-SYSEQU.TEXT
        .INCLUDE TLASM-SYSMACS.TEXT
        .INCLUDE TLASM-SYSERR.TEXT
        .INCLUDE TLASM-QUICKMACS.TEXT
        .INCLUDE TLASM-TOOLMACS.TEXT
        .INCLUDE TLASM-TOOLEQU.TEXT
        .INCLUDE TLASM-RESEQU.TEXT
        .INCLUDE TLASM-FSEQU.TEXT
        .LIST
                •
        .PROC   HDFormat,1

WaitDilog   .EQU    143         ;ID for wait dialog
OKBtn       .EQU    2           ;OK button item # in dialogs

BasDrive    .EQU    4           ;drive # for drive on builtin port
EraseCmd    .EQU    31          ;erase entire disk cmd for driver

BlankDisk   .EQU    0           ;format physical disk
LisaDisk    .EQU    1           ;format logical disk (shared with Lisa OS)
MacDisk     .EQU    2           ;format logical disk (shared with MacWorks)

MaxFDB      .EQU    148         ;max file directory blocks allowed (about 1036 files)

DskType     .EQU    8           ;offset from A6 to disk type param
EraseTyp    .EQU    CSParam+4   ;offset to erase type (all or shared) parameter for erase call
DiskSize    .EQU    20          ;offset into CSParam record for physical size of disk

PhyErase    .EQU    0           ;physical erase, ie. erase complete disk (physical size)
LogErase    .EQU    1           ;logical erase, ie. erase logical disk, using offset

; for local storage

StackBuf    .EQU    -4                  ; save for buffer pointer
IOPBlk      .EQU    StackBuf-4          ; ptr to parameter block area
MountErr    .EQU    IOPBlk-2            ; save of new mount error
MountDrv    .EQU    MountErr-2          ; drive # we're looking for
DiskType    .EQU    MountDrv-2          ; drive type
DQElPtr     .EQU    DiskType-4          ; drive queue element ptr for this drive
DlogResult  .EQU    DQElPtr-2           ; result from modal dialog
OldPort     .EQU    DlogResult-4        ; save area for caller's grafport


Start       LINK    A6,#OldPort         ; reserve space for locals, save A6
            MOVEM.L D3-D7/A2-A5,-(SP)   ; preserve registers

; get space for a parameter block

            MOVEQ   #127,D0
@0          CLR.L   -(SP)               ; clear a buffer area on the stack
            DBRA    D0,@0
            MOVE.L  SP,StackBuf(A6)     ; buffer for init, wrdir calls

            MOVEQ   #(IOVQElSize/2)-1,D0
@1          CLR.W   -(SP)               ; clear a parameter block on the stack
            DBRA    D0,@1
```

```
            MOVE.L  SP,IOPBlk(A6)       ; IOPBlk for IO calls
            CLR     MountErr(A6)        ; no error yet
            MOVEQ   #BasDrive,D1        ; drive we're looking for
            MOVE    D1,MountDrv(A6)     ; save it

            MOVEQ   #NSDrvErr,D3        ; assume we can't find it in the drive q
            MOVE.L  DrvQHdr+QHead,D0    ; find the drive queue element
@2          BEQ     HDIExit             ; exit if drive not found in drive queue
            MOVE.L  D0,A2
            CMP.W   DQDrive(A2),D1      ; this entry?
            BEQ.S   @3                  ; br if so
            MOVE.L  QLink(A2),D0
            BRA.S   @2
@3          MOVE.L  A2,DQElPtr(A6)      ; save ptr

            MOVE.L  DQDrive(A2),IODrvNum(SP) ; fill in IODrvNum, IORefNum

;  setup for doing dialog

FirstDialog
            CLR.L   D2
            CMP.W   #BlankDisk,DskType(A6)  ; is the request to completely zero the disk?
            BNE.S   @1                  ; nope, use logical disk size (ie. physical size - offset)

            MOVE.L  IOPBLK(A6),A0       ; set up IO ptr
            MOVE.W  #DrvStsCode,CSCode(A0)  ; get real size of disk
            _Status
            LEA     CSParam(A0),A0      ; get ptr to returned values
            MOVE.W  DiskSize(A0),D0     ; get physical (actual) size of disk
            BRA.S   @2

@1          MOVE.W  DQDrvSize(A2),D0    ; get logical drive size (physical size - offset)
@2          MOVEQ   #90,D1              ; format at about 90 blocks per second
            DIVU    D1,D0               ; compute total seconds
            MOVE.W  D0,D2               ; save it
            CLR.L   D0
            MOVEQ   #60,D1              ; convert to minutes, seconds
            DIVU    D1,D2
            MOVE.W  D2,D0               ; save minutes in D0
            CLR     D2
            SWAP    D2                  ; round seconds next higher
            ADDQ    #5,D2               ;  multiple of 10
            MOVEQ   #10,D1
            DIVU    D1,D2
            MULU    D1,D2               ; D2 now has seconds

            MOVE.L  StackBuf(A6),A0     ; use stack buffer for string conversion
            CLR.W   -(SP)               ; call _NumToString to convert minutes
            _Pack7
            MOVE.L  A0,A1               ; A1 has ptr to minutes string

            MOVE.L  D2,D0               ; get seconds
            MOVEQ   #16,D1              ; offset buffer ptr
            ADD.L   D1,A0
            CLR.W   -(SP)               ; use _NumToString
```

```
            _Pack7                      ; A0 gets ptr to seconds string

            MOVEM.L A0/A1,-(SP)         ; setup minutes, seconds string
            CLR.L   -(SP)               ; NIL for parms 3 and 4
            CLR.L   -(SP)
            _ParamText                  ; voila!

; now allocate the dialog

            CLR.L   -(SP)               ; space for result
            MOVE.W  #WaitDilog,-(SP)    ; our unique ID
            CLR.L   -(SP)               ; use heap storage
            MOVE.L  MinusOne,-(SP)      ; put it up in front
            _GetNewDialog

            MOVE.L  (SP), A4            ; get new dialog pointer, leave for SetPort
            PEA     oldPort(A6)         ; save oldPort
            _GetPort
            _SetPort                    ; and start writing in dialog window

FmtDialog
            MOVE.L  IOPBlk(A6),A0       ; get ptr to param blk
            MOVE.W  #HDDrive,IOVRefNum(A0)      ; set up volume number
            MOVE.W  #HDrfnum,IORefnum(A0)       ; set up driver refnum
            _UnMountVol                 ; unmount old drive (flush the volume)
                                        ; ignore errors (already unmounted, no such volume, etc)

@1          MOVE.L  A4,-(SP)            ; get dialog ptr
            _DrawDialog                 ; and draw dialog
            LEA     MyFilter,A0         ; filter proc passes back on null
            BSR     GoDlog              ; call ModalDialog

            BSR     Format              ; try formatting and zeroing it (error returned in D3)

            MOVE.L  oldPort(A6), -(SP)  ; push old port
            _SetPort                    ; restore it
            MOVE.L  A4, -(SP)           ; get rid of dialog and item list
            _DisposDialog

; exit format...error passed in D3, cleans up stack and unlinks A6, then returns

HDIExit     MOVE.W  D3,D0
            ADD     #(512+IOVQElSize),SP ; clean up stack space . . .
            MOVEM.L (SP)+,D3-D7/A2-A5    ; restore regs
            UNLK    A6
            MOVE.L  (SP)+,A0             ; get return address
            ADDQ    #2,SP                ; strip off parms
            MOVE.W  D0,(SP)              ; save result
            JMP     (A0)                 ; return to caller . . .


GoDlog      MOVE.L  A0,-(SP)            ; save filter proc ptr on stack
            PEA     DlogResult(A6)      ; code-sharing routine
            _ModalDialog
            MOVE.W  DlogResult(A6),D0
            RTS
```

```
;*****************************************************************************
; here's the filter proc

MyFilter
          MOVEM.L (SP)+,D1-D2/A0-A1   ; D1-return; D2-item var
                                      ; A0-event add; A1-dialog add
          CLR     (SP)                ; assume failure return
          TST.W   EvtNum(A0)          ; null event?
          BNE.S   FPExit              ; then return any old true

FPTrue
          ADDQ.B  #1,(SP)             ; turn false into true
FPExit
          MOVE.L  D1,A1               ; get return address
          JMP     (A1)                ; return


;_____________________________________________________________________
;
; Routine:      Format
; Arguments:      D0.W (output) -- result code (0 if correctly formatted)
; Function:     This routine does a control call to the driver to let it
;               format and zero the device as appropriate.
;_____________________________________________________________________

Format    MOVEM.L D2-D7/A0-A6,-(SP)   ; preserve all registers

          MOVE.L  IOPBlk(A6),A0
          MOVE.W  #EraseCmd,CSCode(A0)     ; disk driver erase command
          MOVE.L  StackBuf(A6),CSParam(A0) ; set buffer ptr
          MOVE.W  DskType(A6),EraseTyp(A0) ; set erase type (<> 0 => logical erase)
@1        _Control                         ; result returned in D0 via IODONE

FmtExit   MOVEM.L (SP)+,A0-A6/D2-D7   ; restore all registers
          MOVE.W  D0,D3               ; set CCR
          RTS

          .END
```

```
; File: HDName.Text
;_____________________________________________________________________
;
; Hard Disk Name/write directory
;
;    written by Rich Castro
;    adapted from Disk Initialization Package by Larry Kenyon
;
; A dialog is presented to the user to name
;  the disk and a directory is then written.  The disk is then mounted as a new
;  device.
;
;  Assumption made that hard disk driver installed before this program
;  is called.
;
;  Stack usage as follows (as offset from stack ptr after LINK):
;
;       8      Function result
;       4      Return address
;       0      Old A6
;      -4      Stack Buffer ptr
;      -8      Parameter block ptr
;      -10     Save of new mount error
;      -12     Save of drive #
;      -16     Drive queue element ptr
;      -18     Dialog result code
;      -22     Save of caller's grafport
;      -534    Buffer area for writing directory
;      -598    Parameter block area
;
;
; Modification History:
;
;  7 Mar 84  RDC  Write initial version
;  9 Mar 84  RDC  Move driver open/mount code to HDOpen routine
; 12 Mar 84  RDC  Make into a Pascal callable routine
; 20 Mar 84  RDC  Change directory initialization parameters
; 26 Mar 84  RDC  Remove _Eject calls which don't make sense for hard disk, add
;                 change of cursor displayed as needed
;  7 Apr 84  RDC  Add fix for bus error caused by InitCursor call
;  9 Apr 84  RDC  Change allocation block size to 4K (8 blocks)
; 11 May 84  RDC  Add support for reinitialize of Mac disk
; 15 May 84  RDC  Add support for reinitialize of Lisa disk (erase cmd)
;            RDC  Increase max file directory size to 148 blocks (1036 files)
; 16 May 84  RDC  Add drive type parameter
; 24 May 84  RDC  Add fix for directory calculations
; 29 May 84  RDC  Add second fix for write directory routine to allow 1 block MDB
;  6 Jun 84  RDC  Delete initialize dialogs; moved to HDUtil for easier manipulation
;  7 Jun 84  RDC  Add time calculation for format operation
; 18 Jun 84  RDC  Add set of watch cursor while directory being written
;  9 Nov 84  KWK  Added setup of vol/driver refnums before UnMountVol/Control calls
; 17 Nov 84  KWK  Stripped out of HD/Format code, removed err/drivetype params, etc.
;
;_____________________________________________________________________

        .NOLIST
```

```
        .INCLUDE TLASM-SYSEQU.TEXT
        .INCLUDE TLASM-SYSMACS.TEXT
        .INCLUDE TLASM-SYSERR.TEXT
        .INCLUDE TLASM-QUICKMACS.TEXT
        .INCLUDE TLASM-TOOLMACS.TEXT
        .INCLUDE TLASM-TOOLEQU.TEXT
        .INCLUDE TLASM-RESEQU.TEXT
        .INCLUDE TLASM-FSEQU.TEXT
        .LIST

        .PROC   HDname,1

NameDilog   .EQU    142         ;ID for getting name for hard disk
NameItem    .EQU    4           ;Item # for default hard disk name
OKBtn       .EQU    2           ;OK button item # in dialogs

ChEnter     .EQU    $03         ;Ascii for 'ENTER' char
ChCR        .EQU    $0D         ;'CR' char
ChColon     .EQU    $3A         ;':' char

BasDrive    .EQU    4           ;drive # for drive on builtin port

MaxFDB      .EQU    148         ;max file directory blocks allowed (about 1036 files)

; for local storage

StackBuf    .EQU    -4                  ; save for buffer pointer
IOPBlk      .EQU    StackBuf-4          ; ptr to parameter block area
MountErr    .EQU    IOPBlk-2            ; save of new mount error
MountDrv    .EQU    MountErr-2          ; drive # we're looking for
DiskType    .EQU    MountDrv-2          ; drive type
DQElPtr     .EQU    DiskType-4          ; drive queue element ptr for this drive
DlogResult  .EQU    DQElPtr-2           ; result from modal dialog
OldPort     .EQU    DlogResult-4        ; save area for caller's grafport


Start       LINK    A6,#OldPort         ; reserve space for locals, save A6
            MOVEM.L D3-D7/A2-A5,-(SP)   ; preserve registers

; get space for a parameter block

            MOVEQ   #127,D0
@0          CLR.L   -(SP)               ; clear a buffer area on the stack
            DBRA    D0,@0
            MOVE.L  SP,StackBuf(A6)     ; buffer for init, wrdir calls

            MOVEQ   #(IOVQElSize/2)-1,D0
@1          CLR.W   -(SP)               ; clear a parameter block on the stack
            DBRA    D0,@1

            MOVE.L  SP,IOPBlk(A6)       ; IOPBlk for IO calls
            CLR     MountErr(A6)        ; no error yet
            MOVEQ   #BasDrive,D1        ; drive we're looking for
            MOVE    D1,MountDrv(A6)     ; save it

            MOVEQ   #NSDrvErr,D3        ; assume we can't find it in the drive q
```

```
            MOVE.L  DrvQHdr+QHead,D0    ; find the drive queue element
@2          BEQ     HDIExit             ; exit if drive not found in drive queue
            MOVE.L  D0,A2
            CMP.W   DQDrive(A2),D1      ; this entry?
            BEQ.S   @3                  ; br if so
            MOVE.L  QLink(A2),D0
            BRA.S   @2
@3          MOVE.L  A2,DQElPtr(A6)      ; save ptr

            MOVE.L  DQDrive(A2),IODrvNum(SP) ; fill in IODrvNum, IORefNum

NameDialog
            CLR.L   -(SP)               ; space for result
            MOVE.W  #NameDilog,-(SP)    ; get naming dialog
            CLR.L   -(SP)               ; use heap storage
            MOVE.L  MinusOne,-(SP)      ; put it up in front
            _GetNewDialog
            MOVE.L  (SP),A4             ; save handle

            MOVE.W  #Nameitem,-(SP)     ; and select current text
            CLR.W   -(SP)               ; startsel
            MOVE.W  #100,-(SP)          ; endsel
            _SelIText

@1          _InitCursor                 ; restore arrow cursor
            LEA     NameFP,A0           ; strip colons, etc.
            BSR.S   GoDlog              ; call ModalDialog
            SUBQ.W  #OKBtn,D0           ; ok?
            BNE.S   @1                  ; loop until ok

            MOVE.L  A4,-(SP)            ; dialog ptr
            MOVE.W  #Nameitem,-(SP)     ; item number for the edit text
            PEA     theType             ; var: type
            PEA     theItem             ; var: itemHandle
            PEA     theRect             ; var: box rectangle
            _GetDItem

            MOVE.L  theItem,-(SP)
            MOVE.L  StackBuf(A6),A2     ; use stack buffer for name
            MOVE.L  A2,-(SP)
            _GetIText                   ; get the text

            MOVE.L  StackBuf(A6),A0     ; use stack buffer for name
            BSR     WrBlnkDir           ; lay down a blank directory . . .

TryMount
            MOVE.L  IOPBlk(A6),A0
            _MountVol                   ; mount it before we exit . . .
            MOVE.W  D0,D3               ; check error (should mount . . .)

RemoveDlg
            MOVE.L  oldPort(A6), -(SP)  ; push old port
            _SetPort                    ; restore it
            MOVE.L  A4, -(SP)           ; get rid of dialog and item list
            _DisposDialog
```

```
HDIExit     MOVE.W  D3,D0
            ADD     #(512+IOVQElSize),SP ; clean up stack space . . .
            MOVEM.L (SP)+,D3-D7/A2-A5    ; restore regs
            UNLK    A6
            MOVE.L  (SP)+,A0             ; get return address
            ADDQ    #4,SP                ; strip off parms
            MOVE.W  D0,(SP)              ; save result
            JMP     (A0)                 ; return to caller . . .


GoDlog      MOVE.L  A0,-(SP)            ; save filter proc ptr on stack
            PEA     DlogResult(A6)      ; code-sharing routine
            _ModalDialog
            MOVE.W  DlogResult(A6),D0
            RTS

;***************************************************************************
; here's the filter proc

NameFP      MOVEQ   #2,D0               ; filter colons, control keys out,
                                        ;  return on CR, Enter
MyFilter
            MOVEM.L (SP)+,D1-D2/A0-A1   ; D1-return; D2-item var
                                        ; A0-event add; A1-dialog add
            CLR     (SP)                ; assume failure return

@0          CMP     #KeyDwnEvt,EvtNum(A0)   ; See if key event type
            BNE.S   FPExit

            MOVE.B  EvtMessage+3(A0),D0 ; get Ascii
            CMP.B   #ChColon,D0         ; colon?
            BNE.S   @1                  ; we don't allow colons in a disk name
            CLR.W   EvtNum(A0)          ; so turn it into a null event

@1          CMP.B   #CHEnter,D0         ; get event message
            BEQ.S   @2                  ; yahoo if enter
            CMP.B   #CHCR,D0            ; get event message
            BNE.S   FPExit              ; exit if not enter or return
@2
            MOVE.L  D2,A0               ; point to item result
            MOVE.W  #OKBtn,(A0)         ; return OK button item

FPTrue
            ADDQ.B  #1,(SP)             ; turn false into true
FPExit
            MOVE.L  D1,A1               ; get return address
            JMP     (A1)                ; return


;_____________________________________________________________
;
; Routine:      WrBlnkDir
; Arguments:      A0.L (input)  -- ptr to volume name for new disk
; Function:     This routine writes a blank master directory on a hard disk.
;               The file directory is zeroed.
;
```

```
; DS  = disk size in 512-byte blocks (given)
; BBS = boot block size = 2
; MDB2 = copy of MDB = 2
; ABBS = allocation blk blk size = 8 (4K)
; AB  = allocation blks =  (DS-BBS-MDB-MDB2-FDB)/ABBS = (DS-2-MDB-2-FDB)/ABBS
; MDB = master directory blks = 1 + (DS/ABBS * 12)/4096 + 1(if rem)3584)
; FDB = file directory blocks = 64
; BBS = boot block size = 2
; DSB = dir start block = 2 + MDB
; CS  = clump size = 8 * ABS
; ASB = allocation start blk = DSB + FDB
; ABS = allocation block size = ABBS * 512
;
;           .Byte   $D2,$D7             ; blank file directory
;           .Long   0,0                 ; creation date, backup date
;           .Word   0                   ; volume attributes
;           .Word   0                   ; number of files
;
;           .Word   DSB                 ; directory start block
;           .Word   FDB                 ; length of directory in 512-byte blks
;           .Word   AB                  ; total number of allocation blocks
;           .Long   ABS                 ; allocation block size
;           .Long   CS                  ; clump size
;           .Word   ASB                 ; allocation block start
;           .Long   $00000001           ; next free file number
;           .Word   AB                  ; number of free allocation blocks
;_____________________________________________________________________


DefltName   .Byte   9
            .Ascii  'Hard Disk'
            .Align  2

WrBlnkDir   MOVEM.L D2-D5/A0-A3,-(SP)   ; save regs other than D0

            CLR.L   TagData+2
            CLR.L   TagData+6           ; master directory has 0-tags

            MOVE.L  StackBuf(A6),A2     ; 512-byte stack buffer
            MOVE.L  A2,A1
            ADD     #DrVN,A1            ; dest for volume name

            MOVE.L  A0,D0               ; name ptr nil?
            BEQ.S   @0                  ; br if so (use default)
            TST.B   (A0)                ; name length zero?
            BNE.S   @1                  ; use default if so

@0          LEA     DefltName,A0

@1          CMP.B   #27,(A0)            ; 27-byte name max
            BLS.S   @2
            MOVE.B  #27,(A0)            ; truncate it

@2          MOVE.B  (A0),D0             ; name length
@3          CMP.B   #ChColon,(A0)       ; colon?
            BNE.S   @4                  ; skip if not
```

```
            MOVE.B  #$20,(A0)           ; replace with a space if so
@4          MOVE.B  (A0)+,(A1)+         ; transfer the name
            SUBQ.B  #1,D0
            BPL.S   @3

            MOVE.L  A2,A0
            ADD     #512,A0             ; buffer's end

@5          CLR.B   (A1)+               ; zero the rest of the buffer
            CMP.L   A0,A1
            BLT.S   @5

FigDiskSize
            MOVEQ   #0,D3               ; zero high word for later
            MOVE.L  DQE1Ptr(A6),A1      ; point to drive queue element
            MOVE.W  DQDrvSize(A1),D3    ; get size from driveq entry

SetMstrInfo
            MOVE.L  A2,A0               ; stack buffer
            MOVE.W  #$D2D7,(A0)+        ; signature word
            MOVE.L  Time,(A0)+          ; creation date
            MOVE.L  Time,(A0)+          ; last backup date
            CLR.L   (A0)+               ; volume attributes, number of files

;  Set directory parameters

            MOVE.L  D3,D2               ; disk size in 512-byte blocks

            MOVE.L  D3,D0               ; use to calculate FDB
            LSR.L   #6,D0               ; DS/64
            MOVE.L  D0,D1
            ADDQ    #1,D1
            BCLR    #0,D1               ; FDB = DS/64 (make it even, though)
            CMP.L   #MaxFDB,D1          ; check against max
            BLS.S   @1                  ; skip if OK
            MOVE.L  #MaxFDB,D1          ; else set FDB to max

@1          MOVEQ   #8,D0               ; ABBS = 8 (4K)

            MOVE.L  D3,D4               ; get disk size
            DIVU    D0,D4               ; DS/ABBS
            SWAP    D4                  ; clear remainder
            CLR     D4
            SWAP    D4
            LSL.L   #2,D4               ; *4
            MOVE.L  D4,D5
            ADD.L   D5,D4               ; *8
            ADD.L   D5,D4               ; *12 (bits required for block map)
            MOVE.L  #4096,D5            ; bits per block
            DIVU    D5,D4
            MOVEQ   #1,D5               ; first MDB block
            ADD.W   D4,D5               ; add quotient
            SWAP    D4                  ; check remainder
            CMP.W   #3584,D4            ; >bits avail in block 1 of MDB?
            BLS.S   @2                  ; skip if OK
            ADDQ.L  #1,D5               ; else incr MDB count
```

```
;@2         CMPI.L  #2,D5              ; check if less than minimum MDB size
;           BGE.S   @3                 ; skip if OK
;           MOVEQ   #2,D5              ; else set to minimum size
@2         MOVE.L  D5,D4              ; get MDB
           ADDQ.L  #2,D4              ; set DSB (offset by 2 boot blocks)
           MOVE.W  D4,(A0)+           ; save directory start block

           SUBQ.L  #4,D2
           SUB.L   D5,D2
           SUB.L   D1,D2
           DIVU    D0,D2              ; AB = DS-4-MDB-FDB / ABBS

           MOVE.W  D1,(A0)+           ; FDB = length of dir in (512-byte) blocks
           MOVE.W  D2,(A0)+           ; AB =  number of alloc blocks this volume

           LSL.L   #8,D0              ; x 512 for alloc blk byte size
           ADD.L   D0,D0
           MOVE.L  D0,(A0)+           ; allocate in these byte quantities
           LSL.L   #3,D0
           MOVE.L  D0,(A0)+           ; num of bytes to try to alloc as a clump

           ADD     D4,D1              ; ASB = FDB + DSB
           MOVE.W  D1,(A0)+           ; starting diskette (512-byte) block in map

           MOVE.L  #$00000001,(A0)+   ; next free file number is 1
           MOVE.W  D2,(A0)            ; all alloc blocks are free now

WrMstrInfo
           MOVE.L  IOPBlk(A6),A0      ; point to general I/O param block
           MOVE.L  A2,IOBuffer(A0)
           MOVE.L  #$00000200,IOByteCount(A0)
           MOVE.L  #$00000400,IOPosOffset(A0) ; absolute address $400
           MOVE.W  #1,IOPosMode(A0)   ; position mode 1 (from disk start)
           _Write                     ; write main directory
           BNE.S   @3                 ; exit on write errors

           SUBQ.L  #2,D3              ; disk size in blks - 2
           LSL.L   #8,D3
           ADD.L   D3,D3              ; x 512 for byte pos of second MDB
           MOVE.L  D3,IOPosOffset(A0) ; put a copy at the end
           _Write
           BNE.S   @3                 ; exit on write errors

           MOVE.L  A2,A1              ; ptr to 512-byte stack buffer
           MOVEQ   #127,D0            ; zero the block
@1         CLR.L   (A1)+
           DBRA    D0,@1

           MOVE.L  #$00000600,IOPosOffset(A0)  ; absolute address $600
@2         _Write                              ; zero the other master directory blocks
           BNE.S   @3
           SUBQ    #1,D5                       ; loop until done or error
           BGT.S   @2

           SUB.L   D4,D1                       ; FDB = ASB - DSB (zero this many dir blks)
           CLR.W   IOPosMode(A0)               ; just zero sequential blocks
```

```
@4          _Write
            BNE.S   @3
            SUBQ    #1,D1                   ; loop until done or error
            BGT.S   @4

@3          MOVEM.L (SP)+,D2-D5/A0-A3   ; restore regs
            MOVE.W  D0,D3
            RTS

;---------------------------------------------------------------------------
;  special data area
;---------------------------------------------------------------------------

theType     .Word   0                   ; item type
theItem     .Long   0                   ; item handle
theRect     .Long   0,0                 ; item rectangle

            .END
```

```
;
;       FILENAME: MWD-HD/WdgtBootDrvr
;
;       Block 0 boot code for Monitor system booting from a Widget
;        7-Nov-84       Stole from Rich Page
;        8-Nov-84       Stripped out non-boot code
;        9-Nov-84       Faked all $A's as header for block 0
;       10-Nov-84       Now reads headers AFTER data, added MOVEQ/ADDQ's
;       11-Nov-84       Moved $AAAA's stuff to WidgetBootBlks so .PROC creates correct offsets
;                       for PC-relative branches
;       12-Nov-84       $AAAA header stuff now handled by Pascal code
;       21-Feb-85  RDC  Change setting of VIA DDRB register so DIAG line (bit 6) is input
;

; PIA REGISTERS for Widget disk interface

ACR     EQU     $58                     ; aux control
PCR     EQU     $60                     ; peripheral control
IFR     EQU     $68                     ; int flags
IER     EQU     $70                     ; int enable
NHS     EQU     $78                     ; reg A no hand shake
DSKBLK  EQU     512                     ; number of bytes in a block
BLKSIZE EQU     255                     ; words-1 in Widget block
PHDRSIZ EQU     9                       ; words-1 in Widget header

; OFFSET EQUS for dskread and dskwrt routines

IOCMD   EQU     -4
IODRV   EQU     -3
BLOCKL  EQU     -2
BLOCKH  EQU     -1

; OFFSET equates for Widget read and write routines

PCMND   EQU     -6
BLKH    EQU     -5
BLKM    EQU     -4
BLKL    EQU     -3
RETRY   EQU     -2
THRESH  EQU     -1

;  BOOT BLOCK ZERO STUFF

WdgtDrvr
        MOVE    #$2700,SR               ; First long on disk = $46FC2700
        LEA     WidgetBoot,A7           ; stack grows down from start of loaded code (block 0)
        MOVEQ   #8,D1                   ; Both block count & address of exception
        MOVE.L  D1,A2
        MOVE.L  #$FCD901,A4             ; Lisa base address
        MOVEQ   #0,D0
        MOVE.L  (A2),A1
        LEA     BUSERR,A0               ; Set up exception handler
        MOVE.L  A0,(A2)
        TST.L   $400000                 ; Read ROM on GLM, Bus Error on Lisa
        MOVE.L  $198,A4                 ; Get GLM base address
        MOVEQ   #1,D0
```

```
BUSERR  MOVE.L  A1,(A2)
        MOVE.B  D0,$14C             ; Set GLM Flag (0=LISA,1=GLM)

        MOVE.B  #$0A,PCR(A4)        ; set ctrl CA2 pulse mode strobe
        MOVE.B  #$00,DDRA(A4)       ; set port A bits to input
        MOVE.B  #$18,ORB(A4)        ; en=true, dir=in, cmd=false
;RDC
        MOVE.B  #$3C,DDRB(A4)       ; set port B bits 0,1,6,7=in, 2,3,4,5=out
;RDC

        MOVE.L  #LDRLOC,A3          ; A3 = PTR TO LOC FOR LOADING BLKS 1..7
        MOVEQ   #0,D3
        MOVE.L  D1,-(A7)            ; COUNT := 8

@1      MOVE.L  A7,D1
        CLR.W   -(A7)               ; RC := 0
        MOVE.L  A7,D0
        MOVE.L  A4,-(A7)            ; BASEADDRESS
        MOVE.L  D0,-(A7)            ; @RC
        MOVE.W  #1,-(A7)            ; DRIVE
        MOVE.L  D1,-(A7)            ; @COUNT
        MOVE.L  D3,-(A7)            ; BLKNUMBER
        MOVE.L  A3,-(A7)            ; @BUFFER
        BSR     PDSKRD

        TST.W   (A7)+               ; RC = 0?
        BNE.S   @3
        ADD.W   #512,A3             ; bump @buffer by block size
        ADDQ.L  #1,D3               ; bump blocknumber by 1
        TST.L   (A7)                ; COUNT = 0 ?
        BNE.S   @1

        JMP     -7*512(A3)          ; jump to first byte of block 1

@3      LEA     ERRMSG,A3
        SUB.L   A2,A2
        MOVEQ   #23,D0
        JMP     ROMEPT


FINDD2  MOVE.B  #$08,ORB(A0)        ; en=true, dir=in, cmd=true
        MOVE.B  #$00,DDRA(A0)       ; set port A bits to input
WFB1    BTST    #1,ORB(A0)          ; wait for busy
        BNE.S   WFB1
        MOVE.B  IRA(A0),D1          ; get port A in D1
        MOVEQ   #0,D0
        CMP.B   D2,D1               ; did pippin return state requested ?
        BNE.S   SNDR1
        MOVEQ   #$55,D0
SNDR1   CLR.B   ORB(A0)             ; #$00 => en=true, dir=out, cmd=true
        ST      DDRA(A0)            ; #$FF => set port A bits to output
        MOVE.B  D0,ORA(A0)          ; send reply 00 or 55
        MOVE.B  #$10,ORB(A0)        ; en=true, dir=out, cmd=false
WFNB1   BTST    #1,ORB(A0)          ; wait for not busy
        BEQ.S   WFNB1
        CLR.B   DDRA(A0)            ; #$00 => set port A bits to input
```

```
        MOVE.B  #$18,ORB(A0)            ; en=true, dir=in, cmd=false
        TST.B   D0                      ; SET CC HERE TO SHARE CODE
        RTS

STAT01
        MOVEQ   #1,D2                   ; try to find state 01
        BSR.S   FINDD2                  ; if state 01 was found then
        BNE.S   COPY6                   ; go send command bytes else
        BSR.S   FINDD2                  ; try again, if state 01 not found then
        BEQ     PDSKERR                 ; return disk error

COPY6
        MOVE.B  #$10,ORB(A0)            ; en=true, dir=out, cmd=false
        ST      DDRA(A0)                ; #$FF => set port A bits to output
        LEA     -6(A6),A1
        MOVEQ   #5,D2                   ; loop six times
COPYSIX MOVE.B  (A1),NHS(A0)
        MOVE.B  (A1)+,ORA(A0)
        DBRA    D2,COPYSIX
        RTS

; Note: Widget controller does auto-mapping to 5:1 interleave, so block
; numbers map directly to sector numbers.

STRTRD  MOVE.L  22(A6),A0               ; set RC to zero
        CLR.W   (A0)
        MOVE.L  26(A6),A0               ; get base address
        CLR.B   PCMND(A6)               ; set command to read
        MOVE.B  13(A6),BLKH(A6)         ; set block number
        MOVE.B  14(A6),BLKM(A6)
        MOVE.B  15(A6),BLKL(A6)
        MOVE.B  #10,RETRY(A6)           ; set retry count
        MOVE.B  #4,THRESH(A6)           ; set threshhold
        BSR.S   STAT01                  ; get 01 byte and send read command
        MOVEQ   #2,D2                   ; get 02 byte
        BSR.S   FINDD2                  ; disk error if not in read state
        BEQ     PDSKERR

        MOVE.B  IRA(A0),-4(A6)          ; get pippin status
        MOVE.B  IRA(A0),-3(A6)
        MOVE.B  IRA(A0),-2(A6)
        MOVE.B  IRA(A0),-1(A6)
        RTS

;       PROCEDURE PDSKRD  (BASEADDR:LONGINT;
;                          VAR RC:INTEGER;
;                          DRIVE:INTEGER;
;                          VAR COUNT:LONGINT;
;                          BLKNUMBER:LONGINT;
;                          VAR BUFFER);
;
;       Stack:
;
;               26      base address
;               22      @RC             ptr to word 0..255
;               20..21  drive           word
```

```
;               16      @Count          ptr to long
;               12..15  Block Number    long
;               8       @Buffer
;               4       Return Address
;               0       Old A6
;               -6      Command Buffer
;               -8      Header flag
;               -28     Header Buffer

PDSKRD  MOVEQ   #1,D0                   ; headers := true
        BRA.S   LDSKRD

NDSKRD  MOVEQ   #0,D0                   ; headers := false
LDSKRD  LINK    A6,#-28

        MOVE.L  12(A6),D2               ; cheap check for Block Number = $FFFFFF
        ROL.L   #8,D2
        BPL.S   @1
        MOVEQ   #0,D0                   ; headers := false

@1      MOVE.W  D0,-8(A6)
        MOVE.L  16(A6),A0               ; get @count
        SUBQ.L  #1,(A0)                 ; decrement count
        BSR.S   STRTRD                  ; try read first time
        TST.W   -2(A6)
        BPL.S   RDNRES
        BSR.S   STRTRD                  ; try read second time

RDNRES
        MOVEQ   #3,D2           ;SET ERROR VALUE
        TST.L   -4(A6)
        BNE.S   PDSKERR         ;WAS ERROR, EXIT

        MOVEQ   #0,D1           ;INIT CHKSUM
        LEA     IRA(A0),A0      ;GET PTR TO DISK IN PORT
        MOVE.L  8(A6),A1        ;get address of BUFFER
        MOVE.W  #BLKSIZE,D2

; now read in the data bytes

READLP
        MOVE.B  (A0),D0         ;GET BYTE FROM DISK
        EOR.B   D0,D1           ;INCLUDE IN RUNNING CHECKSUM
        MOVE.B  D0,(A1)+        ;AND STORE IT IN BUFFER
        MOVE.B  (A0),D0         ;GET BYTE FROM DISK
        EOR.B   D0,D1           ;INCLUDE IN RUNNING CHECKSUM
        MOVE.B  D0,(A1)+        ;AND STORE IT IN BUFFER
        DBRA    D2,READLP       ;REPEAT UNTIL DONE

; now read in the header

        TST.W   -8(A6)
        BEQ.S   FINISH          ; no headers, don't do chksum verification

        LEA     -28(A6),A1      ; set ptr to header buffer
        MOVEQ   #0,D2           ; clear out count
```

```
        MOVEQ   #PHDRSIZ,D2     ; set header words - 1

READHDR
        MOVE.B  (A0),D0         ;GET BYTE FROM DISK
        EOR.B   D0,D1           ;INCLUDE IN RUNNING CHECKSUM
        MOVE.B  D0,(A1)+        ;AND STORE IT IN BUFFER
        MOVE.B  (A0),D0         ;GET BYTE FROM DISK
        EOR.B   D0,D1           ;INCLUDE IN RUNNING CHECKSUM
        MOVE.B  D0,(A1)+        ;AND STORE IT IN BUFFER
        DBRA    D2,READHDR      ;REPEAT UNTIL DONE

        TST.B   D1              ;CHECKSUM = 0, GOOD READ
        BEQ.S   FINISH
        TST.B   -22(A6)         ;SPECIAL HEADER FLAG, IGNORE CHKSUM
        BPL.S   FINISH

        MOVEQ   #4,D2           ;READ ERROR, SET ERROR NUMBER

PDSKERR MOVE.L  22(A6),A0       ; get return code ptr
        NEG.B   D2              ; error is 128..255
        MOVE.W  D2,(A0)         ; set return code

FINISH  UNLK    A6              ;
        MOVE.L  (A7)+,A0        ;GET RETURN ADDRESS
        ADD.W   #22,A7          ;STRIP PARAMS
        JMP     (A0)            ;AND RETURN

; Pad it out to 512 bytes.  The last eight bytes are set by the Widget
; driver (since they actually are the last eight bytes of the header
; block). The first 7 are 0's, w/the last byte = the block checksum.  I
; don't need these bytes anyway, so it just doesn't matter, it just
; doesn't matter, it just doesn't matter...

;       |   20 bytes  |     492 bytes     |   12 bytes  | 7 bytes | 1 byte   |
;       +-------------------------------------------------------------------+
;       | Fake header | Block 0 boot code | Header info |  Zeroed | Checksum |
;       +-------------------------------------------------------------------+

ERRMSG  .ASCII  'ERROR '
        .WORD   0

        .BLOCK  54,0
```

```
;
;       FILENAME: MWD-HD/WidgetBootBlks.text
;
;       Modified:
;
;        6-Nov-84       Ken Krugler -- Hacked to work w/MacWorks
;        8-Nov-84       Added 20 $A's hack to front of boot blocks
;       11-Nov-84       Shift block 0 so PC-relative offsets are correct (otherwise off by $14
;                       because of fake header block)
;       (23Apr85)  RDC  Add support for square pixel screen
;                       Remove initialization of screen to white (leave boot ROM desktop)
;
;       Memory Map:
;
;       high mem        +----------------------+
;                       +                      +
;                       +  Upper Screen Memory +
;                       +                      +
;                       +----------------------+ ($160) = ($174)
;                       +                      +
;                       +  Lower Screen Memory +
;                       +                      +
;                       +----------------------+ ($110) = ($170)
;                       +                      +
;                       +   Monitor Load Area  +
;                       +                      +
;                       +----------------------+ ($10C)
;                       +                      +
;                       +         Free         +
;                       +                      +
;                       +----------------------+
;                       +                      +
;                       +  Loader read by Blk0 +
;                       +                      +
;                       +----------------------+ $20800
;                       +                      +
;                       + Initial Screen Memory +
;                       +                      +
;                       +----------------------+ $10000
;                       +                      +
;                       +   Directory Buffer   +
;                       +                      +
;       stack base      +----------------------+ $10000
;                       +                      +
;                       +   Stack and Globals  +
;                       +                      +
;                       +----------------------+
;                       +                      +
;                       +                      +
;                       +                      +
;       low mem         +----------------------+
;
;
;

        .PROC   WidgetBoot,0
```

```
;
; PIA REGISTERS for Widget disk interface

ORB     EQU     0                       ; output regs
ORA     EQU     8
IRA     EQU     ORA                     ; input regs
IRB     EQU     ORB
DDRB    EQU     $10                     ; data direction regs
DDRA    EQU     $18

ROMEPT  EQU     $FE0084         ; entry point for ROM monitor
CpuROMid EQU    $FE3FFC         ; Lisa CPU ROM version id (3 = square pixels)          (23Apr85)
LDRLOC  EQU     $20800          ; load pt for full loader at $20800
LOMEM   EQU     $02A4           ; low memory address
HIMEM   EQU     $0294           ; high memory address
TMSBASE EQU     $D00-40         ; the MSBASE for the system
STKBASE EQU     $10000          ; Address of base of stack
ADDRSYM EQU     $2000           ; Address for monitor symbols
VIDLATCH EQU    $FCE800         ; address of the video latch
COPSDRB EQU     4               ; Dir Reg B
FDIR    EQU     4               ; Bit number for Floppy Disk Int Request
COPSIFR EQU     $1A             ; Interrupt flag register
PORTA   EQU     2               ; Offset to Port A

CRTROW  EQU     0               ;BYTE, ROW FOR SIMULATED CRT
CRTCOL  EQU     CRTROW+1        ;BYTE, COLUMN FOR SIMULATED CRT
CRTSTAT EQU     CRTCOL+1        ;BYTE, STATE OF CRT SIMULATOR
CTLSFLG EQU     CRTSTAT+1       ;BYTE, UNUSED

BUFSZ   EQU     24576
BUFBK   EQU     BUFSZ/512
BLKSZ   EQU     512
MSGSZ   EQU     2048
DIRSZ   EQU     2048
DIR     EQU     0               ; A5+ (growing directory upwards)
BUF     EQU     -BUFSZ          ; object file buffer, MUST be first !
MSGS    EQU     BUF-MSGSZ       ; message buffer for prtname
BLK     EQU     MSGS-BLKSZ      ; single block for CONFIG and BOOTFILES
LOADANY EQU     BLK-2           ; indicates a file was loaded
LOADSYM EQU     LOADANY-2       ; load monitor symbols flag
PRTINFO EQU     LOADSYM-2       ; print info flag
MSBASE  EQU     PRTINFO-4       ; globals for character printing

ROWBYTS EQU     MSBASE-2        ; bytes for each scan line
ROW8BTS EQU     ROWBYTS-2       ; bytes to offset to 8th scan line
RBYTES  EQU     ROW8BTS-2       ; bytes for each chr row
RLONGS  EQU     RBYTES-2        ; long-words for each chr row
COPSVIA EQU     RLONGS-4        ; address of via for the COPS
BASEADR EQU     COPSVIA-4       ; address of pia for the WIDGET
BLKOFS  EQU     BASEADR-2       ; offset of root volume
PAINTW  EQU     BLKOFS-2        ; paint screen white flag
LASTGLB EQU     PAINTW

; directory entry equates
```

```
FSTBLK  EQU     0               ; dir entry for DFIRSTBLK
LSTBLK  EQU     2               ; dir entry for DLSTBLK
FKIND   EQU     4               ; dir entry for file kind, status

; case fkind = securdir or untypedfile

DVID    EQU     6               ; dir entry for title field
DEOVBLK EQU     14              ; dir entry for end of volume field
DNUMFLS EQU     16              ; dir entry for number files
DLOADTM EQU     18
DLASTBT EQU     20              ; most recent date setting

; case fkind = normal files

DTID    EQU     6               ; dir entry for title field
LSTBYTE EQU     22              ; dir entry for lastbyte
DACCESS EQU     24              ; dir entry for date
DELENG  EQU     26              ; length in bytes of dir entry

FIXCONTRAST     .EQU    1       ; try to fix the contrast

; BOOT BLOCK ZERO CODE

; Include the block 0 boot driver code

        .INCLUDE MWD-HD/WdgtBootDrvr.TEXT

; START OF SECONDARY LOADER (begins at Block 1)

TPLUS512                                ; WidgetBoot + 512 bytes !!

; shift block 0 into the correct position

        LEA     WidgetBoot,A0           ; A0 = ptr to 1st byte of real code
        MOVE.L  A0,A1                   ;
        SUB.W   #20,A1                  ; A1 = ptr to where code should be
        MOVEQ   #0,D0                   ; clear out count
        MOVE.W  #122,D0                 ; move 123 longs = 492 bytes = 512 - 20 byte header

@0      MOVE.L  (A0)+,(A1)+             ; move a byte down
        DBRA    D0,@0                   ; loop till done

; try to reset contrast value here, since it got screwed up somehow.
; A4 still points to VIA1 (hard disk/parallel printer port)
; remember to adjust padding at end of code

        .IF     FIXCONTRAST
        BSET    #2,DDRB(A4)             ; DEN/ = output
        BSET    #2,ORB(A4)              ; DEN/ = 1 (disk disabled)
        MOVE.B  #$FF,DDRA(A4)           ; Port A = output
        MOVE.B  #$80,ORA(A4)            ; write out data (some reasonable start contrast)

        BSET    #7,DDRB(A4)             ; program WCNT as an output
        BCLR    #7,ORB(A4)              ; yank the line down
        BSET    #7,ORB(A4)              ; yank up
        BCLR    #7,DDRB(A4)             ; program WCNT as an input
```

```
        MOVE.B  #$00,DDRA(A4)           ; Port A = input
        BCLR    #2,ORB(A4)              ; DEN/ = 0 (disk enabled)
        .ENDC


; Get on with the business of booting

        MOVE    #$2700,SR               ; turn off interrupts
        MOVE.L  #STKBASE,A7
        LINK    A5,#LASTGLB
        MOVE.W  #0,PAINTW(A5)           ; don't change screen                   <23Apr85>
        CLR.W   MSGS(A5)
        CLR.W   LOADANY(A5)
        CLR.W   LOADSYM(A5)
        CLR.W   PRTINFO(A5)

        TST.B   $14C
        BEQ.S   @5
        MOVE.L  #$18000,D0              ; GLM default screen locations
        MOVE.L  D0,$110
        MOVE.L  D0,$160
        MOVE.L  #$CC0000,D0
        MOVE.L  D0,$170
        MOVE.L  D0,$174
        MOVEQ   #74,D0                  ; GLM row bytes
        MOVE.L  #$D00001,COPSVIA(A5)    ; VIA address for Keyboard and Timers
        MOVE.L  $198,BASEADR(A5)        ; Get base address for boot WIDGET
        BRA.S   @6

@5      MOVE.L  #$18000,D0              ; LISA default screen locations
        MOVE.L  D0,$110
        MOVE.L  D0,$160
        MOVE.L  D0,$170
        MOVE.L  D0,$174

        CMP.B   #3,CpuROMid             ; check for square pixel Lisa           <23Apr85>
        BNE.S   @3                      ; skip if not                           <23Apr85>
        MOVEQ   #76,D0                  ; square pixel row bytes                <23Apr85>
        BRA.S   @4                      ;                                       <23Apr85>

@3      MOVEQ   #90,D0                  ; LISA row bytes
@4      MOVE.L  #$FCDD81,COPSVIA(A5)    ; Via 2 base for NEW & OLD I/O boards
        MOVE.L  #$FCD901,BASEADR(A5)    ; Base address for WIDGET (built-in)

@6      MOVE.W  D0,ROWBYTS(A5)
        MOVE.W  D0,D1
        LSL.W   #3,D1
        MOVE.W  D1,ROW8BTS(A5)
        MULU    #10,D0
        MOVE.W  D0,RBYTES(A5)
        LSR.W   #2,D0
        MOVE.W  D0,RLONGS(A5)
        BSR     SETVIDP

        MOVE.L  #LDRLOC+$E54,A0         ; get start of boot volume block number
```

```
        MOVE.W  (A0),D0
        LSL.W   #3,D0
        MOVE.W  D0,BLKOFS(A5)

        PEA     DIR(A5)
        MOVE.W  #2,-(A7)
        MOVE.W  #4,-(A7)
        BSR     READBLKS                ; Read the directory

        LEA     DIR(A5),A0
        CMP.W   #42,2(A0)
        BNE.S   @2
        MOVE.W  DNUMFLS(A0),D0
        ADD.W   #1,D0
        MULS    #DELENG,D0
        SUB.W   #DIRSZ,D0
        BMI.S   @2
        ADD.W   #511,D0
        LSR.W   #8,D0
        LSR.W   #1,D0
        CMP.W   #36,D0
        BLT.S   @1
        MOVE.W  #36,D0
@1      PEA     DIRSZ(A0)
        MOVE.W  #6,-(A7)
        MOVE.W  D0,-(A7)
        BSR     READBLKS                ; Read the directory
@2      CLR.W   -(A7)
        PEA     CONFIGF
        MOVE.W  #1,-(A7)
        PEA     DIR(A5)
        BSR     DIRSRCH
        TST.W   (A7)
        BNE.S   FCONFIG
        PEA     CONFIGF
        BRA     ERROR
FCONFIG LEA     DIR(A5),A0
        ADD.W   (A7)+,A0
        PEA     BLK(A5)
        MOVE.W  (A0),-(A7)
        MOVE.W  #1,-(A7)
        BSR     READBLKS                ; Read the CONFIG.DATA file
        LEA     BLK(A5),A0
        ADD.W   #6,A0
@1      MOVE.W  (A0)+,D0                ; Get next count word
        BEQ.S   DATAEND
        ROR.W   #8,D0
        MOVE.L  (A0)+,D1                ; Get next address word
        ROR.W   #8,D1
        SWAP    D1
        ROR.W   #8,D1
        SWAP    D1
        MOVE.L  D1,A1
@2      MOVE.B  (A0)+,(A1)+
        SUB.W   #1,D0                   ; Copy each data byte
        BNE.S   @2
```

```
        BRA.S   @1
DATAEND TST.W   BLK(A5)
        BEQ.S   @1
        MOVE.L  HIMEM,D0
        MOVE.L  LOMEM,D1
        SUB.L   D1,D0           ; memory size
        MOVE.L  D0,D2
        SUB.L   #1,D0
        MOVE.L  D0,$114         ; memory top
        SUB.L   #$7FFF,D0
        MOVE.L  D0,$160         ; upper screen base
        MOVE.L  D0,$174
        SUB.L   #$8000,D0
        MOVE.L  D0,$110         ; lower screen base
        MOVE.L  D0,$170
        ASR.L   #1,D2
        MOVE.L  D2,$13C         ; default stack ptr
@1      BSR     SETVIDP
        CLR.L   $118            ; APPLE not available
        MOVE.L  $110,$10C
        PEA     SCREENB
        BSR     PRTNAME
        SUB.L   #$1000,$10C     ; code buffer
        PEA     DBGDATA
        BSR     PRTNAME
        CLR.W   -(A7)
        PEA     BOOTFLS
        MOVE.W  #1,-(A7)
        PEA     DIR(A5)
        BSR     DIRSRCH
        TST.W   (A7)
        BNE.S   @2
        PEA     BOOTFLS
        BRA     ERROR
@2      LEA     DIR(A5),A0
        ADD.W   (A7)+,A0
        PEA     BLK(A5)
        MOVE.W  (A0),-(A7)
        MOVE.W  #1,-(A7)
        BSR     READBLKS                ; Read the BOOTFILES.DATA file
        PEA     BLK(A5)
NEXTF   MOVE.L  (A7),A0
        TST.B   (A0)
        BEQ.S   LOADMON
        CLR.W   -(A7)
        SUB.B   #$20,(A0)
        MOVE.L  A0,-(A7)
        MOVE.W  #1,-(A7)
        PEA     DIR(A5)
        BSR     DIRSRCH
        TST.W   (A7)
        BNE.S   @1
        MOVE.W  #4,D0
        TST.W   (A7)+                   ; to expose address of filename
        BRA     ERROR
@1      LEA     DIR(A5),A0
```

```
        ADD.W   (A7)+,A0
        MOVE.L  A0,-(A7)
        BSR     LOADIT                  ; Load the next object file
        MOVE.L  A5,-(A7)
        MOVE.L  A7,STKBASE
        MOVE.L  $10C,A0
        JSR     (A0)
        MOVE.L  STKBASE,A7
        MOVE.L  (A7)+,A5
        MOVE.W  #1,LOADANY(A5)
        MOVE.L  (A7),-(A7)
        BSR     PRTNAME
        ADD.L   #16,(A7)
        BRA     NEXTF
LOADMON CLR.W   -(A7)
        PEA     MONITOR
        MOVE.W  #1,-(A7)
        PEA     DIR(A5)
        BSR     DIRSRCH
        TST.W   (A7)
        BNE.S   @1
        PEA     MONITOR
        BRA     ERROR
@1      LEA     DIR(A5),A0
        ADD.W   (A7)+,A0
        MOVE.L  A0,-(A7)
        BSR     LOADIT                  ; Load the Monitor object
        PEA     MONITOR
        BSR     PRTNAME
        TST.W   LOADSYM(A5)
        BEQ.S   JMP2MON
        CLR.W   -(A7)
        PEA     SYMBOLS
        MOVE.W  #1,-(A7)
        PEA     DIR(A5)
        BSR     DIRSRCH
        TST.W   (A7)
        BEQ.S   JMP2MON
        LEA     DIR(A5),A0
        ADD.W   (A7)+,A0
        MOVE.L  A0,-(A7)
        BSR     LMONSYM                 ; Load the Monitor symbols
        PEA     MSG6
        BSR.S   WRTSTR

JMP2MON
        MOVE    SR,-(A7)                ; Reset the Keyboard
        MOVE    #$2700,SR               ; Ints off
        MOVE.L  COPSVIA(A5),A0          ; Cops Base Address
        MOVE.B  COPSDRB(A0),D0          ; Get Dir B reg
        MOVE.B  D0,D1                   ; save it
        OR.B    #1,D0                   ; make bit 0 output
        BCLR    #0,(A0)                 ; set reset signal
        MOVE.B  D0,COPSDRB(A0)          ; make sure of Dir B
        MOVE.L  #5200,D2                ; delay 12ms
WAIT4RS DBF     D2,WAIT4RS
```

```
        BSET    #1,(A0)                 ; remove reset signal
        MOVE.B  D1,COPSDRB(A0)          ; restore Dir B reg
        MOVE    (A7)+,SR                ; Ints on
        MOVE.L  $10C,A0
        JMP     (A0)                    ; Go For It !


WRTSTR  MOVE.L  4(A7),A2
        MOVE.L  (A7)+,(A7)
        CLR.W   D2
        MOVE.B  (A2)+,D2
        BRA     WRITE
;
ERROR   PEA     MSG2
        MOVE.W  #$111E,MSBASE(A5)       ; row=17, col=30
        BSR.S   WRTSTR
ERROR2  BSR.S   WRTSTR
HANG    BRA     HANG
;
SCREENB .BYTE   10
        .ASCII  'SCREENBASE '
;
DBGDATA .BYTE   13
        .ASCII  'DEBUGGER.DATA'
;
CONFIGF .BYTE   11
        .ASCII  'CONFIG.DATA'
;
BOOTFLS .BYTE   14
        .ASCII  'BOOTFILES.DATA '
;
MONITOR .BYTE   11
        .ASCII  'MONITOR.OBJ'
;
SYMBOLS .BYTE   15
        .ASCII  'MONITOR.SYMBOLS'
;
MSG1    .BYTE   15
        .ASCII  'Disk Read Error'
;
MSG2    .BYTE   15
        .ASCII  'Failed to find '
;
MSG3    .BYTE   14
        .ASCII  ' is loaded at  '
;
MSG4    .BYTE   45
        .ASCII  'About to boot, press Mouse Button to continue'
;
MSG5    .BYTE   13,13,13
        .ASCII  'Booting ...'
;
MSG6    .BYTE   24,13
        .ASCII  'Loaded Monitor symbols'
        .BYTE   13,0
;
```

```
PRTSTR  LINK    A6,#0
        TST.W   LOADANY(A5)
        BEQ.S   @1
        TST.W   PRTINFO(A5)
        BEQ.S   @1
        LEA     MSGS(A5),A2
        MOVE.W  (A2)+,D2
        BEQ.S   @0
        BSR     WRITE
        MOVE.W  #0,MSGS(A5)
@0      MOVE.L  8(A6),-(A7)
        BSR.S   WRTSTR
        BRA.S   @3
@1      LEA     MSGS(A5),A0
        MOVE.L  A0,A1
        MOVE.W  (A1)+,D0
        ADD.W   D0,A1
        CLR.W   D1
        MOVE.L  8(A6),A2
        MOVE.B  (A2)+,D1
        ADD.W   D1,D0
        MOVE.W  D0,(A0)
@2      MOVE.B  (A2)+,(A1)+
        SUB.W   #1,D1
        BNE.S   @2
@3      UNLK    A6
        MOVE.L  (A7)+,(A7)
        RTS

PRTNAME MOVE.L  (SP)+,(SP)
        RTS

SETVIDP CLR.L   MSBASE(A5)

        TST.W   PAINTW(A5)
        BEQ.S   @3

        BSR.S   @1
        .BYTE   2,27,$2A,0
@1      BSR     WRTSTR

        TST.B   $14C
        BNE.S   @3
        MOVE.L  $160,D0         ; get screen start
        ADD.L   LOMEM,D0        ; bias this by start of memory
        LSR.L   #8,D0           ; and convert to 32K page
        LSR.L   #7,D0
        MOVE.B  D0,VIDLATCH     ; set the video page latch

@3      RTS
;
FILLBUF PEA     BUF(A5)
        MOVE.W  -2(A6),-(A7)
        MOVE.L  8(A6),A0
        MOVE.W  2(A0),D0
        SUB.W   -2(A6),D0
```

```
        CMP.W   #BUFBK-1,D0
        BHI.S   @1
        MOVE.W  D0,-(A7)
        BRA.S   @2
@1      MOVE.W  #BUFBK,-(A7)
@2      BSR     READBLKS
        ADD.W   #BUFBK,-2(A6)
        LEA     BUF(A5),A0
        RTS
;
;       PROCEDURE LMONSYM(VAR DIRENTRY)
;
;       Stack
;
;       8       Pointer to directory entry for symbols file
;       4       Return address
;       0       Old A6
;       -2      Nextblock
;       -6      4 BYTE VALUE
;       -14     8 CHAR NAME
;       -18     Address for monitor symbols
;       -20     Counter
;
LMONSYM LINK    A6,#-18
        MOVE.L  8(A6),A0
        MOVE.W  (A0),-2(A6)
        BSR     FILLBUF
        MOVE.L  #ADDRSYM,A1
        MOVE.L  A1,-18(A6)
@1      MOVE.W  #6,-20(A6)
        LEA     -14(A6),A2
@4      CMP.L   A0,A5
        BNE.S   @5
        MOVE.L  A1,-(A7)
        BSR     FILLBUF
        MOVE.L  (A7)+,A1
@5      MOVE.W  (A0)+,(A2)+     ; fill -14(A6) thru -4(A6)
        SUB.W   #1,-20(A6)
        BNE.S   @4
        TST.L   -14(A6)
        BNE.S   @2
        TST.L   -10(A6)
        BNE.S   @2
        TST.L   -6(A6)
        BEQ.S   @3
@2      MOVE.L  -14(A6),(A1)+   ; copy name
        MOVE.L  -10(A6),(A1)+
        MOVE.L  -6(A6),D0       ; copy value
        ADD.L   $10C,D0
        MOVE.L  D0,(A1)+
        BRA.S   @1
@3      MOVE.L  -18(A6),$406
        MOVE.L  A1,$40A
        UNLK    A6
        MOVE.L  (A7)+,(A7)
        RTS
```

```
;
;       PROCEDURE LOADIT(VAR DIRENTRY)
;
;       Stack
;
;       8       Pointer to directory entry for this file
;       4       Return address
;       0       Old A6
;       -2      Nextblock
;
LOADIT  LINK    A6,#-2
        MOVE.L  8(A6),A0
        MOVE.W  (A0),-2(A6)
        BSR     FILLBUF
@1      CMP.B   #$85,(A0)
        BEQ.S   @2
        CLR.B   (A0)
        ADD.L   (A0),A0
        BRA.S   @1
@2      CLR.B   (A0)
        MOVE.L  (A0)+,D0
        TST.L   (A0)+
        SUB.L   #8,D0
        MOVE.L  $10C,A1
        SUB.L   D0,A1
        MOVE.L  A1,$10C
@3      CMP.L   A0,A5
        BNE.S   @4
        MOVEM.L D0/A1,-(A7)
        BSR     FILLBUF
        MOVEM.L (A7)+,D0/A1
@4      MOVE.W  (A0)+,(A1)+
        SUB.L   #2,D0
        BNE.S   @3
        UNLK    A6
        MOVE.L  (A7)+,(A7)
        RTS
;
;       DIRSRCH
;
;               18      func result
;               14      @FTID
;               12      FINDPERM
;               8       @FDIR
;               4       Return address
;               0       Old A6
;
DIRSRCH LINK    A6,#0
        MOVE.L  8(A6),A1                ; get ptr to dirbuf
        MOVE.W  #0,18(A6)               ; return 0 as default
        MOVE.L  #DELENG,D3              ; initialize offset
        MOVE.L  14(A6),A0               ; point A0 at the filename
        MOVE.W  DNUMFLS(A1),D1          ; get number of files
        BEQ.S   DIRSRCX                 ; number of files = 0 ?
        ADD.W   #DELENG+DTID,A1         ; point A1 at first entry
DIRSRLP MOVEM.L A3/A4,-(A7)             ; save A3 and A4
```

```
        MOVE.L  A1,A4                   ; A4 is used for title compare
        MOVE.L  A0,A3                   ; A3 is title to look for
        CLR.W   D2
        MOVE.B  (A0),D2                 ; length of name in bytes
@1      CMPM.B  (A3)+,(A4)+             ; check each byte for equal
        BNE.S   @2
        SUB.W   #1,D2                   ; compare length+1 bytes
        BCC.S   @1
        SUB.W   #DTID,A1                ; found it leave A1 at start of entry
        MOVEM.L (A7)+,A3/A4             ; restore A3 and A4
        BRA.S   @4
@2      MOVEM.L (A7)+,A3/A4             ; restore A3 and A4
@3      ADD.W   #DELENG,A1              ; skip to next dir entry
        ADD.W   #DELENG,D3              ; also update offset
        SUB.W   #1,D1                   ; any files left ?
        BNE.S   DIRSRLP
        BRA.S   DIRSRCX                 ; file not found
@4      MOVE.W  12(A6),D0               ; if findperm = daccess.year<>100 then
        MOVE.W  DACCESS(A1),D2
        AND.W   #$FE00,D2
        CMP.W   #$C800,D2
        SNE     D2
        AND.W   #1,D2
        CMP.W   D0,D2
        BEQ.S   @5
        ADD.W   #DTID,A1
        BRA.S   @3
@5      MOVE.W  D3,18(A6)               ;   dirsearch:=offset
DIRSRCX UNLK    A6
        MOVE.L  (A7)+,A0                ; pop return address
        ADD.W   #10,A7                  ; delete parameters
        JMP     (A0)
;
;       PROCEDURE READBLKS(VAR BUFFER; BLOCK,COUNT:INTEGER);
;
;       Stack
;
;       12      Address of buffer
;       10      block
;       8       count
;       4       return address
;       0       old A6
;
READBLKS
        LINK    A6,#0
        MOVE.W  BLKOFS(A5),D0
        ADD.W   D0,10(A6)

@1      TST.W   8(A6)
        BEQ.S   @3

        MOVE    SR,-(A7)
        MOVE    #$2700,SR
        MOVE.L  #1,-(A7)                ; COUNT := 1
        MOVE.L  A7,D1
        CLR.W   -(A7)                   ; RC := 0
```

```
        MOVE.L  A7,D0
        MOVE.L  BASEADR(A5),-(A7)       ; BASE ADDRESS
        MOVE.L  D0,-(A7)                ; @RC
        MOVE.W  #1,-(A7)                ; DRIVE
        MOVE.L  D1,-(A7)                ; @COUNT
        CLR.L   D0
        MOVE.W  10(A6),D0
        MOVE.L  D0,-(A7)                ; BLKNUMBER
        MOVE.L  12(A6),-(A7)            ; @BUFFER
        BSR     PDSKRD
        TST.W   (A7)+
        BEQ.S   @2

        PEA     MSG1
        BRA     ERROR2

@2
        TST.L   (A7)+
        MOVE    (A7)+,SR
        SUB.W   #1,8(A6)
        ADD.W   #1,10(A6)
        ADD.L   #512,12(A6)
        BRA.S   @1

@3      UNLK    A6                      ; flush locals
        MOVE.L  (A7)+,A0                ; get return address
        ADD.W   #8,A7                   ; flush params
        JMP     (A0)                    ; and return

;
;
;
GETCOPS MOVE.L  COPSVIA(A5),A0          ; Get via base address
        MOVE.B  COPSIFR(A0),D0          ; Load interrupt flag register
        BPL.S   GETCOPS
        BTST    #1,D0                   ; Test for COPS interrupt
        BEQ.S   GETCOPS
        MOVE.B  PORTA(A0),D0            ; Get byte from COPS
        RTS
;
;       STATE 3 -- GET RESET CODE
;
WAITR   BSR     GETCOPS                 ; discard reset code
        BRA.S   NOKEY
;
;       STATE 1 -- GET DELTA X
;
WAITX   BSR     GETCOPS                 ; discard delta x
;
;       STATE 2 -- GET DELTA Y
;
WAITY   BSR     GETCOPS                 ; discard delta y
;
;       NO KEY STROKE EXIT
;
NOKEY   CLR.W   D1
```

```
        RTS
;
;       STATE 0 -- GET A MOUSE CODE, RESET CODE, OR KEY STROKE
;
GETKEY  BSR     GETCOPS                 ; Get byte, Mouse ?
        BEQ.S   WAITX
        CMP.B   #$80,D0                 ; Reset code ?
        BEQ.S   WAITR
        MOVE.W  #1,D1
        RTS
;
WRITE
        LEA     MSBASE(A5),A3
        TST     D2                      ; any at all?
        BLE.S   EXIT                    ;  no, just leave
        BSR.S   CLRCUR                  ; remove cursor once per call
WRITE1  MOVE.B  (A2)+,D0                ;  and loop thru buffer
        CMP.B   #$0D,D0                 ; is this a CR ?
        BEQ.S   CRLF                    ;  yes, simulate extra stuff
        CMP.B   #$10,D0                 ; how about a DLE?
        BEQ.S   WRDLE                   ;  yes, it's special too
        BSR     PUTC
WRNEXT  SUBQ    #1,D2
        BGT.S   WRITE1
WREXIT  BSR.S   SETCUR                  ; then, set it on again
        BRA.S   EXIT
CRLF    BSR     PUTC                    ; put out the explicit CR
        MOVE.B  #$0A,D0                 ; then, the implicit LF
        BSR     PUTC
        BRA.S   WRNEXT
WRDLE   MOVE.B  #4,CRTSTAT(A3)
        BRA.S   WRNEXT
EXIT    RTS
;
;       Cursor routines.
;
CLRCUR                                  ; code is the same folks
SETCUR
        BSR.S   SETA1                   ; set A1 with proper byte address
        ADD.W   ROW8BTS(A5),A1
        MOVE.W  #8,D0
@1      SUB.W   ROWBYTS(A5),A1
        NOT.B   (A1)                    ;  then complement it
        SUB.W   #1,D0
        BNE.S   @1
        RTS                             ; and thats all there is...
;
;       SETA1           Sets A1 to proper address based upon
; current values of CRTROW and CRTCOL.
;
SETA1
        CLR     D3                      ; note: we assume D3 is free
        MOVE.B  CRTROW(A3),D3
        MULU    RBYTES(A5),D3           ; D3 = byte offset of "CRT"
        ADD     RBYTES(A5),D3           ;  plus an extra one
        ADD.L   $174,D3                 ; addin real screen address
```

```
        MOVE.L  D3,A1                   ;  and set it up
        CLR     D3
        MOVE.B  CRTCOL(A3),D3
        ADDA    D3,A1                   ; and add in column offset
        RTS
;
;       SCROLL  - move contents of screen up one whole line
; We assume that we are at bottom line when called. CRTCOL will be
; left alone, but CRTROW will be set at 32
;
SCROLL                                  ; entry from anywhere
        MOVE    CRTROW(A3),-(SP)        ; save current COLumn
        MOVE    #$0100,CRTROW(A3)       ;  and set to beginning of first row
        BSR.S   SETA1                   ;  and get address of screen
        MOVE.L  A1,A0                   ; set from ptr also
        ADDA    RBYTES(A5),A0
        MOVE    RLONGS(A5),D1           ; set loop for long copies
        LSL.W   #5,D1                   ; 32*RLONGS
@1      MOVE.L  (A0)+,(A1)+
        SUBQ    #1,D1
        BGT.S   @1
        MOVE    (SP)+,CRTROW(A3)        ; restore old info
        MOVE.B  #32,CRTROW(A3)          ;  but peg at bottom
        RTS
;
;       PUTLF   advance CRTROW; this may cause a scroll if at bottom
;
PUTLF   ADDQ.B  #1,CRTROW(A3)
        CMP.B   #32,CRTROW(A3)
        BLS.S   @9                      ; skip if its ok
        BSR     SCROLL                  ;  else, do a scroll operation
@9      RTS
;
;       PUTVT   move cursor up one row;  peg at top
;
PUTVT   SUBQ.B  #1,CRTROW(A3)
        BGT.S   @9
        MOVE.B  #1,CRTROW(A3)
@9      RTS
;
;       PUTBS   move cursor left one position;
;
PUTBS   SUBQ.B  #1,CRTCOL(A3)
        BGT.S   @9
        MOVE.B  #1,CRTCOL(A3)
@9      RTS
;
;       PUTFF   move cursor right one position;
;
PUTFF   MOVE.W  ROWBYTS(A5),D0
        SUB.W   #2,D0
        ADDQ.B  #1,CRTCOL(A3)
        CMP.B   CRTCOL(A3),D0
        BHI.S   @9
        MOVE.B  D0,CRTCOL(A3)           ; pin at right
@9      RTS
```

```
;
;       PUTSPCL         Handle special characters here; such things
; as cursor controls and ESC non-sense.
;
PUTSPCL                                 ; character is now in D0 (-32)
        ADD.B   #$20,D0                 ; just for niceness.
        CMP.B   #$1B,D0                 ; ESC ?
        BNE.S   @1                      ;  no, skip
        MOVE.B  #1,CRTSTAT(A3)          ; else, set state for next time
        RTS                             ;  and exit
@1      CMP.B   #$1E,D0                 ; RS
        BNE.S   @2
        MOVE    #$0101,CRTROW(A3)       ; set to home (1,1)
        RTS
@2      CMP.B   #$08,D0                 ; BS (left arrow)
        BEQ.S   PUTBS
        CMP.B   #$0C,D0                 ; FF (right arrow)
        BEQ.S   PUTFF
        CMP.B   #$0B,D0                 ; VT (up arrow)
        BEQ.S   PUTVT
        CMP.B   #$0A,D0                 ; LF (down arrow)
        BEQ     PUTLF
        CMP.B   #$0D,D0                 ; CR
        BNE.S   @9
        MOVE.B  #1,CRTCOL(A3)
@9      RTS                             ; none of the above
;
;       special ESC characters here
;
ESCT                                    ; erase to end of line
        MOVE.W  ROWBYTS(A5),D1
        SUB.W   #1,D1                   ; compute number at end
        SUB.B   CRTCOL(A3),D1
        BLE.S   @9
        BSR     SETA1                   ; set screen pointer
@1      ADD.W   ROW8BTS(A5),A1
        MOVE.W  #8,D0
@2      SUB.W   ROWBYTS(A5),A1
        CLR.B   (A1)                    ; CLEAR
        SUB.W   #1,D0
        BNE.S   @2
        ADD.W   #1,A1
        SUBQ    #1,D1
        BGT.S   @1
@9      RTS
;
ESCSTAR                                 ; clear the whole screen here folks
        MOVE.L  $174,A1                 ; get screen start
        MOVE.L  #0,D0                   ; get some white
        TST.B   $14C
        BNE.S   @2
        MOVE.W  #8189,D1                ; 364 lines worth of longs: 364*90/4-1
@1      MOVE.L  D0,(A1)+                ; do next long of screen memory
        DBF     D1,@1
        MOVE.L  #-1,D0                  ; get some black
        MOVE.L  D0,(A1)+                ; do next long of screen memory
```

```
        MOVE.L  D0,(A1)+                ; do last long of screen memory
        BRA.S   @4
@2      MOVE.W  #32767,D1               ; zap total of 128K for GLM
@3      MOVE.L  D0,(A1)+
        DBF     D1,@3
@4      MOVE    #$0101,CRTROW(A3)       ; set starting cursor location
        RTS
;
ESCY                                    ; clear from cursor loc to end of screen
        CMP.W   #$0101,CRTROW(A3)       ; is this at top of screen?
        BEQ.S   ESCSTAR                 ;  yes, do full screen then
        MOVE    CRTROW(A3),-(SP)        ; save current location
        CMP.B   #1,CRTCOL(A3)           ;  is it at left?
        BEQ.S   @1                      ;  yes, save some time
        BSR.S   ESCT                    ; no, clear end of this line
        ADDQ.B  #1,CRTROW(A3)
@1      CLR.B   CRTCOL(A3)              ;  and pretend at start of next line
        MOVE    #33,D1                  ; compute rows to clear
        SUB.B   CRTROW(A3),D1
        BLE.S   @9                      ; skip out if none
        MULU    RLONGS(A5),D1           ; else, compute loop values
        BSR     SETA1                   ;  setup A1 with address
@2      CLR.L   (A1)+                   ; and do it to it
        SUBQ    #1,D1
        BGT.S   @2
@9      MOVE    (SP)+,CRTROW(A3)        ; restore correct cursor location
        RTS
;
;-------------------------------------------------------
;
;       PUTC    The real worker of this whole mess.  On entry, D0 has
; the byte to be output.  We are responsible for putting it out (if
; a valid byte), updating CRT pointers, etc.  We also use CTLSKNT
; as a ctrl-S emulation function.
;
PUTC                                    ; D0 is data
        AND.W   #$7F,D0                 ; make sure of upper stuff
        BEQ.S   @2                      ; NUL doesn't do anything
        CLR.W   D1                      ; handle state simulation
        MOVE.B  CRTSTAT(A3),D1
        LSL     #2,D1                   ; QUAD FOR JUMP INDEX
        JMP     PUTCTBL(D1)
@2      RTS                             ; NULs are totally ignored
;
PUTCTBL JMP     PUTC0
        JMP     PUTC1
        JMP     PUTC2
        JMP     PUTC3
        JMP     PUTC4
;
PUTC0                                   ; state 0, normal stuff
        SUB.B   #$20,D0                 ; check it for graphic symbol
        BLT     PUTSPCL                 ; if special then go do it
        CMP.B   #$5F-$20,D0
        BLE.S   @0
        SUB.B   #$20,D0                 ; make upper case
```

```
@0      LEA     FONTTBL,A0
        BRA.S   @2
@1      ADD.W   #14,A7              ; delete bytes from last loop
@2      MOVE.B  #6,D3               ; get 7 more bytes
@3      MOVE.B  (A0)+,D1            ;   next byte
        MOVE.B  D1,-(A7)
        AND.B   #$FC,(A7)           ;   mask off repeat bits
        AND.B   #3,D1               ;   extract repeat count
        BEQ.S   @5
        SUB.B   D1,D3               ;   account for repeat count
@4      MOVE.B  (A7),-(A7)
        SUB.B   #1,D1               ;   push for each repeat count
        BNE.S   @4
@5      SUB.B   #1,D3               ;   decr counter for next byte
        BPL.S   @3
        SUB.B   #1,D0               ; decr character counter
        BPL.S   @1
        CLR.B   -(A7)
        BSR.S   SETA1               ; get screen ptr to A1
        ADD.W   ROW8BTS(A5),A1
        MOVE.W  #8,D0
@6      SUB.W   ROWBYTS(A5),A1
        MOVE.B  (A7)+,(A1)
        SUB.W   #1,D0
        BNE.S   @6
        BRA     PUTFF               ; share code to advance cursor
;
PUTC1                               ; ESC was just seen
        CMP.B   #$3D,D0             ;  ESC-=  (cursor addressing)
        BNE.S   @1
        MOVE.B  #2,CRTSTAT(A3)      ;  and wait for Y value
        RTS
@1      CLR.B   CRTSTAT(A3)         ; reset state for the rest
        CMP.B   #$54,D0             ; ESC-T, erase to end of line
        BEQ.S   ESCT
        CMP.B   #$59,D0             ; ESC-Y, erase to end of screen
        BEQ.S   ESCY
        CMP.B   #$2A,D0             ; ESC-*, erase screen
        BEQ.S   ESCSTAR
        RTS                         ; none of the above
PUTC2                               ; ESC-= seen, expect row value
        SUB.B   #$1F,D0
        MOVE.B  D0,CRTROW(A3)
        MOVE.B  #3,CRTSTAT(A3)
        RTS
PUTC3                               ; ESC-=, Y, expect column value
        SUB.B   #$1F,D0
        MOVE.B  D0,CRTCOL(A3)
        CLR.B   CRTSTAT(A3)
        CMP.B   #32,CRTROW(A3)      ; make limit checks now
        BLS.S   @1
        MOVE.B  #32,CRTROW(A3)
@1      MOVE.W  ROWBYTS(A5),D0
        SUB.W   #2,D0
        CMP.B   CRTCOL(A3),D0
        BHI.S   @2
```

```
        MOVE.B  D0,CRTCOL(A3)
@2      RTS
PUTC4   CLR.B   CRTSTAT(A3)             ; handles dle expansion
        ANDI    #$7F,D0
        SUB     #$20,D0
        BLE.S   @2
        MOVE    D0,-(SP)
@1      MOVE.B  #$20,D0
        BSR     PUTC
        SUBQ    #1,(SP)
        BGT.S   @1
        ADDQ    #2,SP
@2      RTS
;
FONTTBL                                 ; font table origin here folks
  .BYTE  $00+3,$00+2                    ;  (space)
  .BYTE  $10+3,$00+1,$10                ;  !
  .BYTE  $48+2,$00+3                    ;  "
  .BYTE  $48+1,$FC,$48,$FC,$48+1        ;  #
  .BYTE  $10,$3C,$50,$38,$14,$78,$10    ;  $
  .BYTE  $00,$C4,$C8,$10,$20,$4C,$8C    ;  %
  .BYTE  $60,$90+1,$60,$94,$88,$74      ;  &
  .BYTE  $08,$10,$20,$00+3              ;  '
  .BYTE  $08,$10,$20+2,$10,$08          ;  (
  .BYTE  $40,$20,$10+2,$20,$40          ;  )
  .BYTE  $10,$54,$38,$7C,$38,$54,$10    ;  *
  .BYTE  $00,$10+1,$7C,$10+1,$00        ;  +
  .BYTE  $00+3,$30+1,$60                ;  ,
  .BYTE  $00+2,$FC,$00+2                ;  -
  .BYTE  $00+3,$00,$30+1                ;  .
  .BYTE  $00,$04,$08,$10,$20,$40,$80    ;  /
  .BYTE  $78,$84,$8C,$B4,$C4,$84,$78    ;  0
  .BYTE  $10,$30,$50,$10+2,$7C          ;  1
  .BYTE  $78,$84,$04,$18,$60,$80,$FC    ;  2
  .BYTE  $78,$84,$04,$38,$04,$84,$78    ;  3
  .BYTE  $08,$18,$28,$48,$FC,$08+1      ;  4
  .BYTE  $FC,$80,$F0,$08,$04,$88,$70    ;  5
  .BYTE  $38,$40,$80,$F8,$84+1,$78      ;  6
  .BYTE  $FC,$84,$08,$10,$20+2          ;  7
  .BYTE  $78,$84+1,$78,$84+1,$78        ;  8
  .BYTE  $78,$84+1,$7C,$04,$08,$70      ;  9
  .BYTE  $00+1,$30+1,$00,$30+1          ;  :
  .BYTE  $00,$30+1,$00,$30+1,$60        ;  ;
  .BYTE  $08,$10,$20,$40,$20,$10,$08    ;  <
  .BYTE  $00+1,$F8,$00,$F8,$00+1        ;  =
  .BYTE  $40,$20,$10,$08,$10,$20,$40    ;  >
  .BYTE  $78,$84,$04,$18,$20,$00,$20    ;  ?
  .BYTE  $38,$44,$94,$AC,$98,$40,$3C    ;  @
  .BYTE  $30,$48,$84,$FC,$84+2          ;  A
  .BYTE  $F8,$44+1,$78,$44+1,$F8        ;  B
  .BYTE  $78,$84,$80+2,$84,$78          ;  C
  .BYTE  $F8,$44+3,$44,$F8              ;  D
  .BYTE  $FC,$80+1,$F0,$80+1,$FC        ;  E
  .BYTE  $FC,$80+1,$F0,$80+2            ;  F
  .BYTE  $78,$84,$80,$9C,$84+1,$78      ;  G
  .BYTE  $84+2,$FC,$84+2                ;  H
```

```
  .BYTE  $38,$10+3,$10,$38              ;  I
  .BYTE  $1C,$08+3,$88,$70              ;  J
  .BYTE  $84,$88,$90,$E0,$90,$88,$84    ;  K
  .BYTE  $80+3,$80+1,$FC                ;  L
  .BYTE  $84,$CC,$B4+1,$84+2            ;  M
  .BYTE  $84,$C4,$A4,$94,$8C,$84,$84    ;  N
  .BYTE  $78,$84+3,$84,$78              ;  O
  .BYTE  $F8,$84+1,$F8,$80+2            ;  P
  .BYTE  $78,$84+2,$94,$88,$74          ;  Q
  .BYTE  $F8,$84+1,$F8,$90,$88,$84      ;  R
  .BYTE  $78,$84,$80,$78,$04,$84,$78    ;  S
  .BYTE  $7C,$10+3,$10+1                ;  T
  .BYTE  $84+3,$84+1,$78                ;  U
  .BYTE  $84+2,$48+1,$30+1              ;  V
  .BYTE  $84+2,$B4+1,$CC,$84            ;  W
  .BYTE  $84+1,$48,$30,$48,$84+1        ;  X
  .BYTE  $44+2,$38,$10+2                ;  Y
  .BYTE  $FC,$04,$08,$30,$40,$80,$FC    ;  Z
  .BYTE  $78,$40+3,$40,$78              ;  [
  .BYTE  $00,$80,$40,$20,$10,$08,$04    ;  \
  .BYTE  $78,$08+3,$08,$78              ;  ]
  .BYTE  $10,$28,$44,$00+3              ;  ^
  .BYTE  $00+3,$00+1,$FC                ;  _
  .BYTE  0

;       MUST COME OUT TO 7 BLOCKS!!!!!

        .IF     FIXCONTRAST
        .BLOCK  424,0
        .ELSE
        .BLOCK  474,0
        .ENDC

; Now follows the 1 block of mount table/volume entry information

        .INCLUDE    MWD-HD/MountTable.text      ; dummy mount table/volume entries

        .END
```

```
;
;       FILENAME: MWInstall-MWI/ProBootDrvr
;
;       Block 0 boot code for Monitor system booting from a Profile
;
; Changes:
;
;2-21-85  RDC   Change setting of VIA DDRB register so DIAG line (bit 6) is input
;


; PIA REGISTERS for profile disk interface
;
ORB     EQU     0                       ; output regs
ORA     EQU     8
IRA     EQU     ORA                     ; input regs
IRB     EQU     ORB
DDRB    EQU     $10                     ; data direction regs
DDRA    EQU     $18
ACR     EQU     $58                     ; aux control
PCR     EQU     $60                     ; peripheral control
IFR     EQU     $68                     ; int flags
IER     EQU     $70                     ; int enable
NHS     EQU     $78                     ; reg A no hand shake
CMDSIZE EQU     4                       ; number of bytes in cmd string
DSKBLK  EQU     512                     ; number of bytes in a block
PCMNDSZ EQU     5                       ; bytes-1 in profile cmd string
BLKSIZE EQU     255                     ; words-1 in profile block
PHDRSIZ EQU     9                       ; words-1 in profile header
;
; OFFSET EQUS for dskread and dskwrt routines
;
IOCMD   EQU     -4
IODRV   EQU     -3
BLOCKL  EQU     -2
BLOCKH  EQU     -1
;
; OFFSET equates for profile read and write routines
;
PCMND   EQU     -6
BLKH    EQU     -5
BLKM    EQU     -4
BLKL    EQU     -3
RETRY   EQU     -2
THRESH  EQU     -1

;
;       BOOT BLOCK ZERO STUFF
;
        MOVE    #$2700,SR               ; = $46FC2700, first long on disk
        LEA     ProfileBoot,A7
        MOVEQ   #8,D1                   ; Both block count & address of exception
        MOVE.L  D1,A2
        MOVE.L  #$FCD901,A4             ; Lisa base address
        MOVEQ   #0,D0
```

```
        MOVE.L  (A2),A1
        LEA     BUSERR,A0               ; Set up exception handler
        MOVE.L  A0,(A2)
        TST.L   $400000                 ; Read ROM on GLM, Bus Error on Lisa
        MOVE.L  $198,A4                 ; Get GLM base address
        MOVEQ   #1,D0
BUSERR  MOVE.L  A1,(A2)
        MOVE.B  D0,$14C                 ; Set GLM Flag (0=LISA,1=GLM)

        MOVE.B  #$0A,PCR(A4)            ; set ctrl CA2 pulse mode strobe
        MOVE.B  #$00,DDRA(A4)           ; set port A bits to input
        MOVE.B  #$18,ORB(A4)            ; en=true, dir=in, cmd=false
;RDC
        MOVE.B  #$3C,DDRB(A4)           ; set port B bits 0,1,6,7=in, 2,3,4,5=out
;RDC
        MOVE.L  #LDRLOC,A3
        CLR.L   D3
        MOVE.L  D1,-(A7)                ; COUNT := 8
@1      MOVE.L  A7,D1
        CLR.W   -(A7)                   ; RC := 0
        MOVE.L  A7,D0
        MOVE.L  A4,-(A7)                ; BASEADDRESS
        MOVE.L  D0,-(A7)                ; @RC
        MOVE.W  #1,-(A7)                ; DRIVE
        MOVE.L  D1,-(A7)                ; @COUNT
        MOVE.L  D3,-(A7)                ; BLKNUMBER
        MOVE.L  A3,-(A7)                ; @BUFFER
        BSR     PDSKRD
        TST.W   (A7)+                   ; RC = 0?
        BNE.S   @3
        ADD.W   #512,A3                 ; bump @buffer by block size
        ADD.L   #1,D3                   ; bump blocknumber by 1
        TST.L   (A7)                    ; COUNT = 0 ?
        BNE.S   @1
        JMP     -7*512(A3)
@3      LEA     ERRMSG,A3
        SUB.L   A2,A2
        MOVEQ   #23,D0
        JMP     ROMEPT


;
FINDD2  MOVE.B  #$08,ORB(A0)            ; en=true, dir=in, cmd=true
        MOVE.B  #$00,DDRA(A0)           ; set port A bits to input
WFB1    BTST    #1,ORB(A0)              ; wait for busy
        BNE.S   WFB1
        MOVE.B  IRA(A0),D1              ; get port A in D1
        MOVEQ   #0,D0
        CMP.B   D2,D1                   ; did pippin return state requested ?
        BNE.S   SNDR1
        MOVEQ   #$55,D0
SNDR1   MOVE.B  #$00,ORB(A0)            ; en=true, dir=out, cmd=true
        MOVE.B  #$FF,DDRA(A0)           ; set port A bits to output
        MOVE.B  D0,ORA(A0)              ; send reply 00 or 55
        MOVE.B  #$10,ORB(A0)            ; en=true, dir=out, cmd=false
WFNB1   BTST    #1,ORB(A0)              ; wait for not busy
```

```
        BEQ.S   WFNB1
        MOVE.B  #$00,DDRA(A0)           ; set port A bits to input
        MOVE.B  #$18,ORB(A0)            ; en=true, dir=in, cmd=false
        TST.B   D0                      ; SET CC HERE TO SHARE CODE
        RTS
;
;
;
STAT01  MOVE.L  #1,D2                   ; try to find state 01
        BSR.S   FINDD2                  ; if state 01 was found then
        BNE.S   COPY6                   ; go send command bytes else
        BSR.S   FINDD2                  ; try again, if state 01 not found then
        BEQ     PDSKERR                 ; return disk error
COPY6   MOVE.B  #$10,ORB(A0)            ; en=true, dir=out, cmd=false
        MOVE.B  #$FF,DDRA(A0)           ; set port A bits to output
        LEA     -6(A6),A1
        MOVEQ   #6,D2
COPYSIX MOVE.B  (A1),NHS(A0)
        MOVE.B  (A1)+,ORA(A0)
        SUB.W   #1,D2
        BNE.S   COPYSIX
        RTS


STRTRD  MOVE.L  22(A6),A0               ; set RC to zero
        CLR.W   (A0)
        MOVE.L  26(A6),A0               ; get base address
        MOVE.B  #0,PCMND(A6)            ; set command to read
        MOVE.B  13(A6),BLKH(A6)         ; set block number
        MOVE.B  14(A6),BLKM(A6)
        MOVE.B  15(A6),D0               ; lsb of block number
        MOVE.B  D0,D1
        TST.W   -8(A6)
        BEQ.S   @1
;
;       REMAP
;
        MOVEQ   #-$10,D1                ; = $F0
        AND.B   D0,D1                   ; mask high 4 bits
        AND.W   #$0F,D0                 ; mask low  4 bits
        ADD.B   INTRLV(D0),D1           ; add in remapped low 4 bits
@1      MOVE.B  D1,BLKL(A6)             ; replace block number
        MOVE.B  #10,RETRY(A6)           ; set retry count
        MOVE.B  #4,THRESH(A6)           ; set threshhold
        BSR.S   STAT01                  ; get 01 byte and send read command
        MOVE.L  #2,D2                   ; get 02 byte
        BSR.S   FINDD2                  ; disk error if not in read state
        BEQ     PDSKERR
        MOVE.B  IRA(A0),-4(A6)          ; get pippin status
        MOVE.B  IRA(A0),-3(A6)
        MOVE.B  IRA(A0),-2(A6)
        MOVE.B  IRA(A0),-1(A6)
        RTS
;
;
;
```

```
INTRLV  .BYTE   0,5,10,15,4,9,14,3,8,13,2,7,12,1,6,11    ; 9:1 INTERLEAVE
;
;       PROCEDURE PDSKRD  (BASEADDR:LONGINT;
;                          VAR RC:INTEGER;
;                          DRIVE:INTEGER;
;                          VAR COUNT:LONGINT;
;                          BLKNUMBER:LONGINT;
;                          VAR BUFFER);
;
;       Stack:
;
;               26      base address
;               22      @RC             ptr to word 0..255
;               20..21  drive           word
;               16      @Count          ptr to long
;               12..15  Block Number    long
;               8       @Buffer
;               4       Return Address
;               0       Old A6
;               -6      Command Buffer
;               -8      Header flag
;               -28     Header Buffer

PDSKRD  MOVEQ   #1,D0                   ; headers := true
        BRA.S   LDSKRD
NDSKRD  MOVEQ   #0,D0                   ; headers := false
LDSKRD  LINK    A6,#-28

        MOVE.L  12(A6),D2               ; cheap check for Block Number = $FFFFFF
        ROL.L   #8,D2
        BPL.S   @1
        MOVEQ   #0,D0                   ; headers := false

@1      MOVE.W  D0,-8(A6)
        MOVE.L  16(A6),A0               ; get @count
        SUB.L   #1,(A0)                 ; decrement count
        BSR.S   STRTRD                  ; try read first time
        TST.W   -2(A6)
        BPL.S   RDNRES
        BSR.S   STRTRD                  ; try read second time
RDNRES  MOVE.L  #3,D2
        TST.L   -4(A6)
        BNE.S   PDSKERR
        CLR.B   D1              ;INIT CSUM
        LEA     IRA(A0),A0
        TST.W   -8(A6)
        BEQ.S   RSKPHDR
        LEA     -28(A6),A1
        MOVE.W  #PHDRSIZ,D2
READHDR
        MOVE.B  (A0),D0         ;GET BYTE FROM DISK
        EOR.B   D0,D1           ;INCLUDE IN RUNNING CHECKSUM
        MOVE.B  D0,(A1)+        ;AND STORE IT IN BUFFER
        MOVE.B  (A0),D0         ;GET BYTE FROM DISK
        EOR.B   D0,D1           ;INCLUDE IN RUNNING CHECKSUM
        MOVE.B  D0,(A1)+        ;AND STORE IT IN BUFFER
```

```
        DBF     D2,READHDR      ;REPEAT UNTIL DONE
RSKPHDR
        MOVE.L  8(A6),A1                ; get address of BUFFER
        MOVE.W  #BLKSIZE,D2
READLP                                  ; read the data bytes
        MOVE.B  (A0),D0         ;GET BYTE FROM DISK
        EOR.B   D0,D1           ;INCLUDE IN RUNNING CHECKSUM
        MOVE.B  D0,(A1)+        ;AND STORE IT IN BUFFER
        MOVE.B  (A0),D0         ;GET BYTE FROM DISK
        EOR.B   D0,D1           ;INCLUDE IN RUNNING CHECKSUM
        MOVE.B  D0,(A1)+        ;AND STORE IT IN BUFFER
        DBF     D2,READLP       ;REPEAT UNTIL DONE
        TST.B   D1
        BEQ.S   FINISH
        TST.W   -8(A6)
        BEQ.S   FINISH
        TST.B   -22(A6)
        BPL.S   FINISH
        MOVE.L  #4,D2
;
PDSKERR MOVE.L  22(A6),A0               ; set RC
        NEG.B   D2                      ; error is 128..255
        MOVE.W  D2,(A0)
;
FINISH  UNLK    A6
        MOVE.L  (A7)+,A0
        ADD.W   #22,A7
        JMP     (A0)
```

```
;
;       FILENAME: MWINSTALL-MWI/ProfileBootBlks.text
;       Monitor Profile bootblocks for MacWorks
;
;       Modified:
;       10-29-84                by Ken Krugler (removed print/mouse stuff)
;                               NOTE: boot code part of must be 7 block ($E00) long,
;                               so if anything changes, adjust .BLOCK at end
;        5-Nov-84               Cleanup, name changing.
;       (23Apr85)       RDC     Add support for square pixel screen
;                               Remove initialization of screen to white (leave boot ROM desktop)
;
;       Memory Map:
;
;       high mem        +----------------------+
;                       +                      +
;                       +  Upper Screen Memory +
;                       +                      +
;                       +----------------------+ ($160) = ($174)
;                       +                      +
;                       +  Lower Screen Memory +
;                       +                      +
;                       +----------------------+ ($110) = ($170)
;                       +                      +
;                       +   Monitor Load Area  +
;                       +                      +
;                       +----------------------+ ($10C)
;                       +                      +
;                       +         Free         +
;                       +                      +
;                       +----------------------+
;                       +                      +
;                       +  Loader read by Blk0 +
;                       +                      +
;                       +----------------------+ $20800
;                       +                      +
;                       + Initial Screen Memory +
;                       +                      +
;                       +----------------------+ $10000
;                       +                      +
;                       +   Directory Buffer   +
;                       +                      +
;       stack base      +----------------------+ $10000
;                       +                      +
;                       +   Stack and Globals  +
;                       +                      +
;                       +----------------------+
;                       +                      +
;                       +                      +
;                       +                      +
;       low mem         +----------------------+
;
;
;
```

```
        .PROC   ProfileBoot,0

ROMEPT  EQU     $FE0084         ; entry point for ROM monitor
CpuROMid EQU    $FE3FFC         ; Lisa CPU ROM version id (3 = square pixels)          (23Apr85)
LDRLOC  EQU     $20800          ; load pt for full loader at $20800
LOMEM   EQU     $02A4           ; low memory address
HIMEM   EQU     $0294           ; high memory address
TMSBASE EQU     $D00-40         ; the MSBASE for the system
STKBASE EQU     $10000          ; Address of base of stack
ADDRSYM EQU     $2000           ; Address for monitor symbols
VIDLATCH EQU    $FCE800         ; address of the video latch
COPSDRB EQU     4               ; Dir Reg B
FDIR    EQU     4               ; Bit number for Floppy Disk Int Request
COPSIFR EQU     $1A             ; Interrupt flag register
PORTA   EQU     2               ; Offset to Port A
;
CRTROW  EQU     0               ;BYTE, ROW FOR SIMULATED CRT
CRTCOL  EQU     CRTROW+1        ;BYTE, COLUMN FOR SIMULATED CRT
CRTSTAT EQU     CRTCOL+1        ;BYTE, STATE OF CRT SIMULATOR
CTLSFLG EQU     CRTSTAT+1       ;BYTE, UNUSED
;
;
;
BUFSZ   EQU     24576
BUFBK   EQU     BUFSZ/512
BLKSZ   EQU     512
MSGSZ   EQU     2048
DIRSZ   EQU     2048
DIR     EQU     0               ; A5+ (growing directory upwards)
BUF     EQU     -BUFSZ          ; object file buffer, MUST be first !
MSGS    EQU     BUF-MSGSZ       ; message buffer for prtname
BLK     EQU     MSGS-BLKSZ      ; single block for CONFIG and BOOTFILES
LOADANY EQU     BLK-2           ; indicates a file was loaded
LOADSYM EQU     LOADANY-2       ; load monitor symbols flag
PRTINFO EQU     LOADSYM-2       ; print info flag
MSBASE  EQU     PRTINFO-4       ; globals for character printing

ROWBYTS EQU     MSBASE-2        ; bytes for each scan line
ROW8BTS EQU     ROWBYTS-2       ; bytes to offset to 8th scan line
RBYTES  EQU     ROW8BTS-2       ; bytes for each chr row
RLONGS  EQU     RBYTES-2        ; long-words for each chr row
COPSVIA EQU     RLONGS-4        ; address of via for the COPS
BASEADR EQU     COPSVIA-4       ; address of pia for the PROFILE
BLKOFS  EQU     BASEADR-2       ; offset of root volume
PAINTW  EQU     BLKOFS-2        ; paint screen white flag
LASTGLB EQU     PAINTW
;
; directory entry equates
;
FSTBLK  EQU     0               ; dir entry for DFIRSTBLK
LSTBLK  EQU     2               ; dir entry for DLSTBLK
FKIND   EQU     4               ; dir entry for file kind, status
; case fkind = securdir or untypedfile
DVID    EQU     6               ; dir entry for title field
DEOVBLK EQU     14              ; dir entry for end of volume field
DNUMFLS EQU     16              ; dir entry for number files
```

```
DLOADTM EQU     18
DLASTBT EQU     20              ; most recent date setting
; case fkind = normal files
DTID    EQU     6               ; dir entry for title field
LSTBYTE EQU     22              ; dir entry for lastbyte
DACCESS EQU     24              ; dir entry for date
DELENG  EQU     26              ; length in bytes of dir entry


; BOOT BLOCK ZERO CODE

        .INCLUDE MWD-HD/PROBOOTDRVR.TEXT     ; Block 0 boot code

;
ERRMSG  .ASCII  'ERROR '                ; pad out block 0 code to 512 bytes long
        .WORD   0,0

; START OF SECONDARY LOADER

TPLUS512                                ; ProfileBoot + 512 bytes !! (Must be)
        MOVE    #$2700,SR
        MOVE.L  #STKBASE,A7
        LINK    A5,#LASTGLB
        MOVE.W  #0,PAINTW(A5)           ; don't change screen                   <23Apr85>
        MOVE.W  #0,MSGS(A5)
        MOVE.W  #0,LOADANY(A5)
        MOVE.W  #0,LOADSYM(A5)
        MOVE.W  #0,PRTINFO(A5)

        TST.B   $14C
        BEQ.S   @5
        MOVE.L  #$18000,D0              ; GLM default screen locations
        MOVE.L  D0,$110
        MOVE.L  D0,$160
        MOVE.L  #$CC0000,D0
        MOVE.L  D0,$170
        MOVE.L  D0,$174
        MOVEQ   #74,D0                  ; GLM row bytes
        MOVE.L  #$D00001,COPSVIA(A5)    ; VIA address for Keyboard and Timers
        MOVE.L  $198,BASEADR(A5)        ; Get base address for boot PROFILE
        BRA.S   @6

@5      MOVE.L  #$18000,D0              ; LISA default screen locations
        MOVE.L  D0,$110
        MOVE.L  D0,$160
        MOVE.L  D0,$170
        MOVE.L  D0,$174

        CMP.B   #3,CpuROMid             ; check for square pixel Lisa           <23Apr85>
        BNE.S   @3                      ; skip if not                           <23Apr85>
        MOVEQ   #76,D0                  ; square pixel row bytes                <23Apr85>
        BRA.S   @4                      ;                                       <23Apr85>

@3      MOVEQ   #90,D0                  ; LISA row bytes
@4      MOVE.L  #$FCDD81,COPSVIA(A5)    ; Via 2 base for NEW & OLD I/O boards
        MOVE.L  #$FCD901,BASEADR(A5)    ; Base address for PROFILE (built-in)
```

```
@6      MOVE.W  D0,ROWBYTS(A5)
        MOVE.W  D0,D1
        LSL.W   #3,D1
        MOVE.W  D1,ROW8BTS(A5)
        MULU    #10,D0
        MOVE.W  D0,RBYTES(A5)
        LSR.W   #2,D0
        MOVE.W  D0,RLONGS(A5)
        BSR     SETVIDP

        MOVE.L  #LDRLOC+$E54,A0
        MOVE.W  (A0),D0
        LSL.W   #3,D0
        MOVE.W  D0,BLKOFS(A5)

        PEA     DIR(A5)
        MOVE.W  #2,-(A7)
        MOVE.W  #4,-(A7)
        BSR     READBLKS                ; Read the directory

        LEA     DIR(A5),A0
        CMP.W   #42,2(A0)
        BNE.S   @2
        MOVE.W  DNUMFLS(A0),D0
        ADD.W   #1,D0
        MULS    #DELENG,D0
        SUB.W   #DIRSZ,D0
        BMI.S   @2
        ADD.W   #511,D0
        LSR.W   #8,D0
        LSR.W   #1,D0
        CMP.W   #36,D0
        BLT.S   @1
        MOVE.W  #36,D0
@1      PEA     DIRSZ(A0)
        MOVE.W  #6,-(A7)
        MOVE.W  D0,-(A7)
        BSR     READBLKS                ; Read the directory
@2      CLR.W   -(A7)
        PEA     CONFIGF
        MOVE.W  #1,-(A7)
        PEA     DIR(A5)
        BSR     DIRSRCH
        TST.W   (A7)
        BNE.S   FCONFIG
        PEA     CONFIGF
        BRA     ERROR
FCONFIG LEA     DIR(A5),A0
        ADD.W   (A7)+,A0
        PEA     BLK(A5)
        MOVE.W  (A0),-(A7)
        MOVE.W  #1,-(A7)
        BSR     READBLKS                ; Read the CONFIG.DATA file
        LEA     BLK(A5),A0
        ADD.W   #6,A0
@1      MOVE.W  (A0)+,D0                ; Get next count word
```

```
        BEQ.S   DATAEND
        ROR.W   #8,D0
        MOVE.L  (A0)+,D1                ; Get next address word
        ROR.W   #8,D1
        SWAP    D1
        ROR.W   #8,D1
        SWAP    D1
        MOVE.L  D1,A1
@2      MOVE.B  (A0)+,(A1)+
        SUB.W   #1,D0                   ; Copy each data byte
        BNE.S   @2
        BRA.S   @1
DATAEND TST.W   BLK(A5)
        BEQ.S   @1
        MOVE.L  HIMEM,D0
        MOVE.L  LOMEM,D1
        SUB.L   D1,D0           ; memory size
        MOVE.L  D0,D2
        SUB.L   #1,D0
        MOVE.L  D0,$114         ; memory top
        SUB.L   #$7FFF,D0
        MOVE.L  D0,$160         ; upper screen base
        MOVE.L  D0,$174
        SUB.L   #$8000,D0
        MOVE.L  D0,$110         ; lower screen base
        MOVE.L  D0,$170
        ASR.L   #1,D2
        MOVE.L  D2,$13C         ; default stack ptr
@1      BSR     SETVIDP
        CLR.L   $118            ; APPLE not available
        MOVE.L  $110,$10C
        PEA     SCREENB
        BSR     PRTNAME
        SUB.L   #$1000,$10C     ; code buffer
        PEA     DBGDATA
        BSR     PRTNAME
        CLR.W   -(A7)
        PEA     BOOTFLS
        MOVE.W  #1,-(A7)
        PEA     DIR(A5)
        BSR     DIRSRCH
        TST.W   (A7)
        BNE.S   @2
        PEA     BOOTFLS
        BRA     ERROR
@2      LEA     DIR(A5),A0
        ADD.W   (A7)+,A0
        PEA     BLK(A5)
        MOVE.W  (A0),-(A7)
        MOVE.W  #1,-(A7)
        BSR     READBLKS                ; Read the BOOTFILES.DATA file
        PEA     BLK(A5)
NEXTF   MOVE.L  (A7),A0
        TST.B   (A0)
        BEQ.S   LOADMON
        CLR.W   -(A7)
```

```
        SUB.B   #$20,(A0)
        MOVE.L  A0,-(A7)
        MOVE.W  #1,-(A7)
        PEA     DIR(A5)
        BSR     DIRSRCH
        TST.W   (A7)
        BNE.S   @1
        MOVE.W  #4,D0
        TST.W   (A7)+                   ; to expose address of filename
        BRA     ERROR
@1      LEA     DIR(A5),A0
        ADD.W   (A7)+,A0
        MOVE.L  A0,-(A7)
        BSR     LOADIT                  ; Load the next object file
        MOVE.L  A5,-(A7)
        MOVE.L  A7,STKBASE
        MOVE.L  $10C,A0
        JSR     (A0)
        MOVE.L  STKBASE,A7
        MOVE.L  (A7)+,A5
        MOVE.W  #1,LOADANY(A5)
        MOVE.L  (A7),-(A7)
        BSR     PRTNAME
        ADD.L   #16,(A7)
        BRA     NEXTF
LOADMON CLR.W   -(A7)
        PEA     MONITOR
        MOVE.W  #1,-(A7)
        PEA     DIR(A5)
        BSR     DIRSRCH
        TST.W   (A7)
        BNE.S   @1
        PEA     MONITOR
        BRA     ERROR
@1      LEA     DIR(A5),A0
        ADD.W   (A7)+,A0
        MOVE.L  A0,-(A7)
        BSR     LOADIT                  ; Load the Monitor object
        PEA     MONITOR
        BSR     PRTNAME
        TST.W   LOADSYM(A5)
        BEQ.S   JMP2MON
        CLR.W   -(A7)
        PEA     SYMBOLS
        MOVE.W  #1,-(A7)
        PEA     DIR(A5)
        BSR     DIRSRCH
        TST.W   (A7)
        BEQ.S   JMP2MON
        LEA     DIR(A5),A0
        ADD.W   (A7)+,A0
        MOVE.L  A0,-(A7)
        BSR     LMONSYM                 ; Load the Monitor symbols
        PEA     MSG6
        BSR.S   WRTSTR
```

```
JMP2MON BRA.S   @2

@2      MOVE    SR,-(A7)            ; Reset the Keyboard
        MOVE    #$2700,SR           ; Ints off
        MOVE.L  COPSVIA(A5),A0      ; Cops Base Address
        MOVE.B  COPSDRB(A0),D0      ; Get Dir B reg
        MOVE.B  D0,D1               ; save it
        OR.B    #1,D0               ; make bit 0 output
        BCLR    #0,(A0)             ; set reset signal
        MOVE.B  D0,COPSDRB(A0)      ; make sure of Dir B
        MOVE.L  #5200,D2            ; delay 12ms
WAIT4RS DBF     D2,WAIT4RS
        BSET    #1,(A0)             ; remove reset signal
        MOVE.B  D1,COPSDRB(A0)      ; restore Dir B reg
        MOVE    (A7)+,SR            ; Ints on
        MOVE.L  $10C,A0
        JMP     (A0)                ; Go For It !

WRTSTR  MOVE.L  4(A7),A2
        MOVE.L  (A7)+,(A7)
        CLR.W   D2
        MOVE.B  (A2)+,D2
        BRA     WRITE
;
ERROR   PEA     MSG2
        MOVE.W  #$111E,MSBASE(A5)   ; row=17, col=30
        BSR.S   WRTSTR
ERROR2  BSR.S   WRTSTR
HANG    BRA     HANG
;
SCREENB .BYTE   10
        .ASCII  'SCREENBASE '
;
DBGDATA .BYTE   13
        .ASCII  'DEBUGGER.DATA'
;
CONFIGF .BYTE   11
        .ASCII  'CONFIG.DATA'
;
BOOTFLS .BYTE   14
        .ASCII  'BOOTFILES.DATA '
;
MONITOR .BYTE   11
        .ASCII  'MONITOR.OBJ'
;
SYMBOLS .BYTE   15
        .ASCII  'MONITOR.SYMBOLS'
;
MSG1    .BYTE   15
        .ASCII  'Disk Read Error'
;
MSG2    .BYTE   15
        .ASCII  'Failed to find '
;
MSG3    .BYTE   14
        .ASCII  ' is loaded at  '
```

```
;
MSG4    .BYTE   45
        .ASCII  'About to boot, press Mouse Button to continue'
;
MSG5    .BYTE   13,13,13
        .ASCII  'Booting ...'
;
MSG6    .BYTE   24,13
        .ASCII  'Loaded Monitor symbols'
        .BYTE   13,0
;
PRTSTR  LINK    A6,#0
        TST.W   LOADANY(A5)
        BEQ.S   @1
        TST.W   PRTINFO(A5)
        BEQ.S   @1
        LEA     MSGS(A5),A2
        MOVE.W  (A2)+,D2
        BEQ.S   @0
        BSR     WRITE
        MOVE.W  #0,MSGS(A5)
@0      MOVE.L  8(A6),-(A7)
        BSR.S   WRTSTR
        BRA.S   @3
@1      LEA     MSGS(A5),A0
        MOVE.L  A0,A1
        MOVE.W  (A1)+,D0
        ADD.W   D0,A1
        CLR.W   D1
        MOVE.L  8(A6),A2
        MOVE.B  (A2)+,D1
        ADD.W   D1,D0
        MOVE.W  D0,(A0)
@2      MOVE.B  (A2)+,(A1)+
        SUB.W   #1,D1
        BNE.S   @2
@3      UNLK    A6
        MOVE.L  (A7)+,(A7)
        RTS


PRTNAME MOVE.L  (SP)+,(SP)
        RTS


SETVIDP CLR.L   MSBASE(A5)

        TST.W   PAINTW(A5)
        BEQ.S   @3

        BSR.S   @1
        .BYTE   2,27,$2A,0
@1      BSR     WRTSTR

        TST.B   $14C
        BNE.S   @3
```

```
        MOVE.L  $160,D0         ; get screen start
        ADD.L   LOMEM,D0        ; bias this by start of memory
        LSR.L   #8,D0           ; and convert to 32k page
        LSR.L   #7,D0
        MOVE.B  D0,VIDLATCH     ; set the video page latch

@3      RTS
;
FILLBUF PEA     BUF(A5)
        MOVE.W  -2(A6),-(A7)
        MOVE.L  8(A6),A0
        MOVE.W  2(A0),D0
        SUB.W   -2(A6),D0
        CMP.W   #BUFBK-1,D0
        BHI.S   @1
        MOVE.W  D0,-(A7)
        BRA.S   @2
@1      MOVE.W  #BUFBK,-(A7)
@2      BSR     READBLKS
        ADD.W   #BUFBK,-2(A6)
        LEA     BUF(A5),A0
        RTS
;
;       PROCEDURE LMONSYM(VAR DIRENTRY)
;
;       Stack
;
;       8       Pointer to directory entry for symbols file
;       4       Return address
;       0       Old A6
;       -2      Nextblock
;       -6      4 BYTE VALUE
;       -14     8 CHAR NAME
;       -18     Address for monitor symbols
;       -20     Counter
;
LMONSYM LINK    A6,#-18
        MOVE.L  8(A6),A0
        MOVE.W  (A0),-2(A6)
        BSR     FILLBUF
        MOVE.L  #ADDRSYM,A1
        MOVE.L  A1,-18(A6)
@1      MOVE.W  #6,-20(A6)
        LEA     -14(A6),A2
@4      CMP.L   A0,A5
        BNE.S   @5
        MOVE.L  A1,-(A7)
        BSR     FILLBUF
        MOVE.L  (A7)+,A1
@5      MOVE.W  (A0)+,(A2)+     ; fill -14(A6) thru -4(A6)
        SUB.W   #1,-20(A6)
        BNE.S   @4
        TST.L   -14(A6)
        BNE.S   @2
        TST.L   -10(A6)
        BNE.S   @2
```

```
        TST.L   -6(A6)
        BEQ.S   @3
@2      MOVE.L  -14(A6),(A1)+   ; copy name
        MOVE.L  -10(A6),(A1)+
        MOVE.L  -6(A6),D0       ; copy value
        ADD.L   $10C,D0
        MOVE.L  D0,(A1)+
        BRA.S   @1
@3      MOVE.L  -18(A6),$406
        MOVE.L  A1,$40A
        UNLK    A6
        MOVE.L  (A7)+,(A7)
        RTS
;
;       PROCEDURE LOADIT(VAR DIRENTRY)
;
;       Stack
;
;       8       Pointer to directory entry for this file
;       4       Return address
;       0       Old A6
;       -2      Nextblock
;
LOADIT  LINK    A6,#-2
        MOVE.L  8(A6),A0
        MOVE.W  (A0),-2(A6)
        BSR     FILLBUF
@1      CMP.B   #$85,(A0)
        BEQ.S   @2
        CLR.B   (A0)
        ADD.L   (A0),A0
        BRA.S   @1
@2      CLR.B   (A0)
        MOVE.L  (A0)+,D0
        TST.L   (A0)+
        SUB.L   #8,D0
        MOVE.L  $10C,A1
        SUB.L   D0,A1
        MOVE.L  A1,$10C
@3      CMP.L   A0,A5
        BNE.S   @4
        MOVEM.L D0/A1,-(A7)
        BSR     FILLBUF
        MOVEM.L (A7)+,D0/A1
@4      MOVE.W  (A0)+,(A1)+
        SUB.L   #2,D0
        BNE.S   @3
        UNLK    A6
        MOVE.L  (A7)+,(A7)
        RTS
;
;       DIRSRCH
;
;               18      func result
;               14      @FTID
;               12      FINDPERM
```

```
;               8       @FDIR
;               4       Return address
;               0       Old A6
;
DIRSRCH LINK    A6,#0
        MOVE.L  8(A6),A1                ; get ptr to dirbuf
        MOVE.W  #0,18(A6)               ; return 0 as default
        MOVE.L  #DELENG,D3              ; initialize offset
        MOVE.L  14(A6),A0               ; point A0 at the filename
        MOVE.W  DNUMFLS(A1),D1          ; get number of files
        BEQ.S   DIRSRCX                 ; number of files = 0 ?
        ADD.W   #DELENG+DTID,A1         ; point A1 at first entry
DIRSRLP MOVEM.L A3/A4,-(A7)             ; save A3 and A4
        MOVE.L  A1,A4                   ; A4 is used for title compare
        MOVE.L  A0,A3                   ; A3 is title to look for
        CLR.W   D2
        MOVE.B  (A0),D2                 ; length of name in bytes
@1      CMPM.B  (A3)+,(A4)+             ; check each byte for equal
        BNE.S   @2
        SUB.W   #1,D2                   ; compare length+1 bytes
        BCC.S   @1
        SUB.W   #DTID,A1                ; found it leave A1 at start of entry
        MOVEM.L (A7)+,A3/A4             ; restore A3 and A4
        BRA.S   @4
@2      MOVEM.L (A7)+,A3/A4             ; restore A3 and A4
@3      ADD.W   #DELENG,A1              ; skip to next dir entry
        ADD.W   #DELENG,D3              ; also update offset
        SUB.W   #1,D1                   ; any files left ?
        BNE.S   DIRSRLP
        BRA.S   DIRSRCX                 ; file not found
@4      MOVE.W  12(A6),D0               ; if findperm = daccess.year<>100 then
        MOVE.W  DACCESS(A1),D2
        AND.W   #$FE00,D2
        CMP.W   #$C800,D2
        SNE     D2
        AND.W   #1,D2
        CMP.W   D0,D2
        BEQ.S   @5
        ADD.W   #DTID,A1
        BRA.S   @3
@5      MOVE.W  D3,18(A6)               ;   dirsearch:=offset
DIRSRCX UNLK    A6
        MOVE.L  (A7)+,A0                ; pop return address
        ADD.W   #10,A7                  ; delete parameters
        JMP     (A0)
;
;       PROCEDURE READBLKS(VAR BUFFER; BLOCK,COUNT:INTEGER);
;
;       Stack
;
;       12      Address of buffer
;       10      block
;       8       count
;       4       return address
;       0       old A6
;
```

```
READBLKS
        LINK    A6,#0
        MOVE.W  BLKOFS(A5),D0
        ADD.W   D0,10(A6)
@1      TST.W   8(A6)
        BEQ.S   @3
        MOVE    SR,-(A7)
        MOVE    #$2700,SR
        MOVE.L  #1,-(A7)                ; COUNT := 1
        MOVE.L  A7,D1
        CLR.W   -(A7)                   ; RC := 0
        MOVE.L  A7,D0
        MOVE.L  BASEADR(A5),-(A7)       ; BASE ADDRESS
        MOVE.L  D0,-(A7)                ; @RC
        MOVE.W  #1,-(A7)                ; DRIVE
        MOVE.L  D1,-(A7)                ; @COUNT
        CLR.L   D0
        MOVE.W  10(A6),D0
        MOVE.L  D0,-(A7)                ; BLKNUMBER
        MOVE.L  12(A6),-(A7)            ; @BUFFER
        BSR     PDSKRD
        TST.W   (A7)+
        BEQ.S   @2
        PEA     MSG1
        BRA     ERROR2
@2      TST.L   (A7)+
        MOVE    (A7)+,SR
        SUB.W   #1,8(A6)
        ADD.W   #1,10(A6)
        ADD.L   #512,12(A6)
        BRA.S   @1
@3      UNLK    A6
        MOVE.L  (A7)+,A0
        ADD.W   #8,A7
        JMP     (A0)
;
;
;
GETCOPS MOVE.L  COPSVIA(A5),A0          ; Get via base address
        MOVE.B  COPSIFR(A0),D0          ; Load interrupt flag register
        BPL.S   GETCOPS
        BTST    #1,D0                   ; Test for COPS interrupt
        BEQ.S   GETCOPS
        MOVE.B  PORTA(A0),D0            ; Get byte from COPS
        RTS
;
;       STATE 3 -- GET RESET CODE
;
WAITR   BSR     GETCOPS                 ; discard reset code
        BRA.S   NOKEY
;
;       STATE 1 -- GET DELTA X
;
WAITX   BSR     GETCOPS                 ; discard delta x
;
;       STATE 2 -- GET DELTA Y
```

```
;
WAITY   BSR     GETCOPS                 ; discard delta y
;
;       NO KEY STROKE EXIT
;
NOKEY   CLR.W   D1
        RTS
;
;       STATE 0 -- GET A MOUSE CODE, RESET CODE, OR KEY STROKE
;
GETKEY  BSR     GETCOPS                 ; Get byte, Mouse ?
        BEQ.S   WAITX
        CMP.B   #$80,D0                 ; Reset code ?
        BEQ.S   WAITR
        MOVE.W  #1,D1
        RTS
;
WRITE
        LEA     MSBASE(A5),A3
        TST     D2                      ; any at all?
        BLE.S   EXIT                    ;  no, just leave
        BSR.S   CLRCUR                  ; remove cursor once per call
WRITE1  MOVE.B  (A2)+,D0                ;  and loop thru buffer
        CMP.B   #$0D,D0                 ; is this a CR ?
        BEQ.S   CRLF                    ;  yes, simulate extra stuff
        CMP.B   #$10,D0                 ; how about a DLE?
        BEQ.S   WRDLE                   ;  yes, it's special too
        BSR     PUTC
WRNEXT  SUBQ    #1,D2
        BGT.S   WRITE1
WREXIT  BSR.S   SETCUR                  ; then, set it on again
        BRA.S   EXIT
CRLF    BSR     PUTC                    ; put out the explicit CR
        MOVE.B  #$0A,D0                 ; then, the implicit LF
        BSR     PUTC
        BRA.S   WRNEXT
WRDLE   MOVE.B  #4,CRTSTAT(A3)
        BRA.S   WRNEXT
EXIT    RTS
;
;       Cursor routines.
;
CLRCUR                                  ; code is the same folks
SETCUR
        BSR.S   SETA1                   ; set A1 with proper byte address
        ADD.W   ROW8BTS(A5),A1
        MOVE.W  #8,D0
@1      SUB.W   ROWBYTS(A5),A1
        NOT.B   (A1)                    ;  then complement it
        SUB.W   #1,D0
        BNE.S   @1
        RTS                             ; and thats all there is...
;
;       SETA1           Sets A1 to proper address based upon
; current values of CRTROW and CRTCOL.
;
```

```
SETA1
        CLR     D3                      ; note: we assume D3 is free
        MOVE.B  CRTROW(A3),D3
        MULU    RBYTES(A5),D3           ; D3 = byte offset of "CRT"
        ADD     RBYTES(A5),D3           ;  plus an extra one
        ADD.L   $174,D3                 ; addin real screen address
        MOVE.L  D3,A1                   ;  and set it up
        CLR     D3
        MOVE.B  CRTCOL(A3),D3
        ADDA    D3,A1                   ; and add in column offset
        RTS
;
;       SCROLL  - move contents of screen up one whole line
; We assume that we are at bottom line when called. CRTCOL will be
; left alone, but CRTROW will be set at 32
;
SCROLL                                  ; entry from anywhere
        MOVE    CRTROW(A3),-(SP)        ; save current COLumn
        MOVE    #$0100,CRTROW(A3)       ;  and set to beginning of first row
        BSR.S   SETA1                   ;  and get address of screen
        MOVE.L  A1,A0                   ; set from ptr also
        ADDA    RBYTES(A5),A0
        MOVE    RLONGS(A5),D1           ; set loop for long copies
        LSL.W   #5,D1                   ; 32*RLONGS
@1      MOVE.L  (A0)+,(A1)+
        SUBQ    #1,D1
        BGT.S   @1
        MOVE    (SP)+,CRTROW(A3)        ; restore old info
        MOVE.B  #32,CRTROW(A3)          ;  but peg at bottom
        RTS
;
;       PUTLF   advance CRTROW; this may cause a scroll if at bottom
;
PUTLF   ADDQ.B  #1,CRTROW(A3)
        CMP.B   #32,CRTROW(A3)
        BLS.S   @9                      ; skip if its ok
        BSR     SCROLL                  ;  else, do a scroll operation
@9      RTS
;
;       PUTVT   move cursor up one row;  peg at top
;
PUTVT   SUBQ.B  #1,CRTROW(A3)
        BGT.S   @9
        MOVE.B  #1,CRTROW(A3)
@9      RTS
;
;       PUTBS   move cursor left one position;
;
PUTBS   SUBQ.B  #1,CRTCOL(A3)
        BGT.S   @9
        MOVE.B  #1,CRTCOL(A3)
@9      RTS
;
;       PUTFF   move cursor right one position;
;
PUTFF   MOVE.W  ROWBYTS(A5),D0
```

```
        SUB.W   #2,D0
        ADDQ.B  #1,CRTCOL(A3)
        CMP.B   CRTCOL(A3),D0
        BHI.S   @9
        MOVE.B  D0,CRTCOL(A3)           ; pin at right
@9      RTS
;
;       PUTSPCL         Handle special characters here; such things
; as cursor controls and ESC non-sense.
;
PUTSPCL                                 ; character is now in D0 (-32)
        ADD.B   #$20,D0                 ; just for niceness.
        CMP.B   #$1B,D0                 ; ESC ?
        BNE.S   @1                      ;  no, skip
        MOVE.B  #1,CRTSTAT(A3)          ; else, set state for next time
        RTS                             ;  and exit
@1      CMP.B   #$1E,D0                 ; RS
        BNE.S   @2
        MOVE    #$0101,CRTROW(A3)       ; set to home (1,1)
        RTS
@2      CMP.B   #$08,D0                 ; BS (left arrow)
        BEQ.S   PUTBS
        CMP.B   #$0C,D0                 ; FF (right arrow)
        BEQ.S   PUTFF
        CMP.B   #$0B,D0                 ; VT (up arrow)
        BEQ.S   PUTVT
        CMP.B   #$0A,D0                 ; LF (down arrow)
        BEQ     PUTLF
        CMP.B   #$0D,D0                 ; CR
        BNE.S   @9
        MOVE.B  #1,CRTCOL(A3)
@9      RTS                             ; none of the above
;
;       special ESC characters here
;
ESCT                                    ; erase to end of line
        MOVE.W  ROWBYTS(A5),D1
        SUB.W   #1,D1                   ; compute number at end
        SUB.B   CRTCOL(A3),D1
        BLE.S   @9
        BSR     SETA1                   ; set screen pointer
@1      ADD.W   ROW8BTS(A5),A1
        MOVE.W  #8,D0
@2      SUB.W   ROWBYTS(A5),A1
        CLR.B   (A1)                    ; CLEAR
        SUB.W   #1,D0
        BNE.S   @2
        ADD.W   #1,A1
        SUBQ    #1,D1
        BGT.S   @1
@9      RTS
;
ESCSTAR                                 ; clear the whole screen here folks
        MOVE.L  $174,A1                 ; get screen start
        MOVE.L  #0,D0                   ; get some white
        TST.B   $14C
```

```
        BNE.S   @2
        MOVE.W  #8189,D1                ; 364 lines worth of longs: 364*90/4-1
@1      MOVE.L  D0,(A1)+                ; do next long of screen memory
        DBF     D1,@1
        MOVE.L  #-1,D0                  ; get some black
        MOVE.L  D0,(A1)+                ; do next long of screen memory
        MOVE.L  D0,(A1)+                ; do last long of screen memory
        BRA.S   @4
@2      MOVE.W  #32767,D1               ; zap total of 128K for GLM
@3      MOVE.L  D0,(A1)+
        DBF     D1,@3
@4      MOVE    #$0101,CRTROW(A3)       ; set starting cursor location
        RTS
;
ESCY                                    ; clear from cursor loc to end of screen
        CMP.W   #$0101,CRTROW(A3)       ; is this at top of screen?
        BEQ.S   ESCSTAR                 ;  yes, do full screen then
        MOVE    CRTROW(A3),-(SP)        ; save current location
        CMP.B   #1,CRTCOL(A3)           ;  is it at left?
        BEQ.S   @1                      ;  yes, save some time
        BSR.S   ESCT                    ; no, clear end of this line
        ADDQ.B  #1,CRTROW(A3)
@1      CLR.B   CRTCOL(A3)              ;  and pretend at start of next line
        MOVE    #33,D1                  ; compute rows to clear
        SUB.B   CRTROW(A3),D1
        BLE.S   @9                      ; skip out if none
        MULU    RLONGS(A5),D1           ; else, compute loop values
        BSR     SETA1                   ;  setup A1 with address
@2      CLR.L   (A1)+                   ; and do it to it
        SUBQ    #1,D1
        BGT.S   @2
@9      MOVE    (SP)+,CRTROW(A3)        ; restore correct cursor location
        RTS
;
;-------------------------------------------------------
;
;       PUTC    The real worker of this whole mess.  On entry, D0 has
; the byte to be output.  We are responsible for putting it out (if
; a valid byte), updating CRT pointers, etc.  We also use CTLSKNT
; as a ctrl-S emulation function.
;
PUTC                                    ; D0 is data
        AND.W   #$7F,D0                 ; make sure of upper stuff
        BEQ.S   @2                      ; NUL doesn't do anything
        CLR.W   D1                      ; handle state simulation
        MOVE.B  CRTSTAT(A3),D1
        LSL     #2,D1                   ; QUAD FOR JUMP INDEX
        JMP     PUTCTBL(D1)
@2      RTS                             ; NULs are totally ignored
;
PUTCTBL JMP     PUTC0
        JMP     PUTC1
        JMP     PUTC2
        JMP     PUTC3
        JMP     PUTC4
;
```

```
PUTC0                                   ; state 0, normal stuff
        SUB.B   #$20,D0                 ; check it for graphic symbol
        BLT     PUTSPCL                 ; if special then go do it
        CMP.B   #$5F-$20,D0
        BLE.S   @0
        SUB.B   #$20,D0                 ; make upper case
@0      LEA     FONTTBL,A0
        BRA.S   @2
@1      ADD.W   #14,A7                  ; delete bytes from last loop
@2      MOVE.B  #6,D3                   ; get 7 more bytes
@3      MOVE.B  (A0)+,D1                ;   next byte
        MOVE.B  D1,-(A7)
        AND.B   #$FC,(A7)               ;   mask off repeat bits
        AND.B   #3,D1                   ;   extract repeat count
        BEQ.S   @5
        SUB.B   D1,D3                   ;   account for repeat count
@4      MOVE.B  (A7),-(A7)
        SUB.B   #1,D1                   ;   push for each repeat count
        BNE.S   @4
@5      SUB.B   #1,D3                   ;   decr counter for next byte
        BPL.S   @3
        SUB.B   #1,D0                   ; decr character counter
        BPL.S   @1
        CLR.B   -(A7)
        BSR.S   SETA1                   ; get screen ptr to A1
        ADD.W   ROW8BTS(A5),A1
        MOVE.W  #8,D0
@6      SUB.W   ROWBYTS(A5),A1
        MOVE.B  (A7)+,(A1)
        SUB.W   #1,D0
        BNE.S   @6
        BRA     PUTFF                   ; share code to advance cursor
;
PUTC1                                   ; ESC was just seen
        CMP.B   #$3D,D0                 ;  ESC-=  (cursor addressing)
        BNE.S   @1
        MOVE.B  #2,CRTSTAT(A3)          ;  and wait for Y value
        RTS
@1      CLR.B   CRTSTAT(A3)             ; reset state for the rest
        CMP.B   #$54,D0                 ; ESC-T, erase to end of line
        BEQ.S   ESCT
        CMP.B   #$59,D0                 ; ESC-Y, erase to end of screen
        BEQ.S   ESCY
        CMP.B   #$2A,D0                 ; ESC-*, erase screen
        BEQ.S   ESCSTAR
        RTS                             ; none of the above
PUTC2                                   ; ESC-= seen, expect row value
        SUB.B   #$1F,D0
        MOVE.B  D0,CRTROW(A3)
        MOVE.B  #3,CRTSTAT(A3)
        RTS
PUTC3                                   ; ESC-=, Y, expect column value
        SUB.B   #$1F,D0
        MOVE.B  D0,CRTCOL(A3)
        CLR.B   CRTSTAT(A3)
        CMP.B   #32,CRTROW(A3)          ; make limit checks now
```

```
        BLS.S   @1
        MOVE.B  #32,CRTROW(A3)
@1      MOVE.W  ROWBYTS(A5),D0
        SUB.W   #2,D0
        CMP.B   CRTCOL(A3),D0
        BHI.S   @2
        MOVE.B  D0,CRTCOL(A3)
@2      RTS
PUTC4   CLR.B   CRTSTAT(A3)             ; handles dle expansion
        ANDI    #$7F,D0
        SUB     #$20,D0
        BLE.S   @2
        MOVE    D0,-(SP)
@1      MOVE.B  #$20,D0
        BSR     PUTC
        SUBQ    #1,(SP)
        BGT.S   @1
        ADDQ    #2,SP
@2      RTS
;
FONTTBL                                 ; font table origin here folks
  .BYTE  $00+3,$00+2                    ;  (space)
  .BYTE  $10+3,$00+1,$10                ;  !
  .BYTE  $48+2,$00+3                    ;  "
  .BYTE  $48+1,$FC,$48,$FC,$48+1        ;  #
  .BYTE  $10,$3C,$50,$38,$14,$78,$10    ;  $
  .BYTE  $00,$C4,$C8,$10,$20,$4C,$8C    ;  %
  .BYTE  $60,$90+1,$60,$94,$88,$74      ;  &
  .BYTE  $08,$10,$20,$00+3              ;  '
  .BYTE  $08,$10,$20+2,$10,$08          ;  (
  .BYTE  $40,$20,$10+2,$20,$40          ;  )
  .BYTE  $10,$54,$38,$7C,$38,$54,$10    ;  *
  .BYTE  $00,$10+1,$7C,$10+1,$00        ;  +
  .BYTE  $00+3,$30+1,$60                ;  ,
  .BYTE  $00+2,$FC,$00+2                ;  -
  .BYTE  $00+3,$00,$30+1                ;  .
  .BYTE  $00,$04,$08,$10,$20,$40,$80    ;  /
  .BYTE  $78,$84,$8C,$B4,$C4,$84,$78    ;  0
  .BYTE  $10,$30,$50,$10+2,$7C          ;  1
  .BYTE  $78,$84,$04,$18,$60,$80,$FC    ;  2
  .BYTE  $78,$84,$04,$38,$04,$84,$78    ;  3
  .BYTE  $08,$18,$28,$48,$FC,$08+1      ;  4
  .BYTE  $FC,$80,$F0,$08,$04,$88,$70    ;  5
  .BYTE  $38,$40,$80,$F8,$84+1,$78      ;  6
  .BYTE  $FC,$84,$08,$10,$20+2          ;  7
  .BYTE  $78,$84+1,$78,$84+1,$78        ;  8
  .BYTE  $78,$84+1,$7C,$04,$08,$70      ;  9
  .BYTE  $00+1,$30+1,$00,$30+1          ;  :
  .BYTE  $00,$30+1,$00,$30+1,$60        ;  ;
  .BYTE  $08,$10,$20,$40,$20,$10,$08    ;  <
  .BYTE  $00+1,$F8,$00,$F8,$00+1        ;  =
  .BYTE  $40,$20,$10,$08,$10,$20,$40    ;  >
  .BYTE  $78,$84,$04,$18,$20,$00,$20    ;  ?
  .BYTE  $38,$44,$94,$AC,$98,$40,$3C    ;  @
  .BYTE  $30,$48,$84,$FC,$84+2          ;  A
  .BYTE  $F8,$44+1,$78,$44+1,$F8        ;  B
```

```
  .BYTE  $78,$84,$80+2,$84,$78          ; C
  .BYTE  $F8,$44+3,$44,$F8              ; D
  .BYTE  $FC,$80+1,$F0,$80+1,$FC        ; E
  .BYTE  $FC,$80+1,$F0,$80+2            ; F
  .BYTE  $78,$84,$80,$9C,$84+1,$78      ; G
  .BYTE  $84+2,$FC,$84+2                ; H
  .BYTE  $38,$10+3,$10,$38              ; I
  .BYTE  $1C,$08+3,$88,$70              ; J
  .BYTE  $84,$88,$90,$E0,$90,$88,$84    ; K
  .BYTE  $80+3,$80+1,$FC                ; L
  .BYTE  $84,$CC,$B4+1,$84+2            ; M
  .BYTE  $84,$C4,$A4,$94,$8C,$84,$84    ; N
  .BYTE  $78,$84+3,$84,$78              ; O
  .BYTE  $F8,$84+1,$F8,$80+2            ; P
  .BYTE  $78,$84+2,$94,$88,$74          ; Q
  .BYTE  $F8,$84+1,$F8,$90,$88,$84      ; R
  .BYTE  $78,$84,$80,$78,$04,$84,$78    ; S
  .BYTE  $7C,$10+3,$10+1                ; T
  .BYTE  $84+3,$84+1,$78                ; U
  .BYTE  $84+2,$48+1,$30+1              ; V
  .BYTE  $84+2,$B4+1,$CC,$84            ; W
  .BYTE  $84+1,$48,$30,$48,$84+1        ; X
  .BYTE  $44+2,$38,$10+2                ; Y
  .BYTE  $FC,$04,$08,$30,$40,$80,$FC    ; Z
  .BYTE  $78,$40+3,$40,$78              ; [
  .BYTE  $00,$80,$40,$20,$10,$08,$04    ; \
  .BYTE  $78,$08+3,$08,$78              ; ]
  .BYTE  $10,$28,$44,$00+3              ; ^
  .BYTE  $00+3,$00+1,$FC                ; _
  .BYTE  0

  .BLOCK        494,0                   ; pad to $E00 in size (KwK)                    <23Apr85>

;
;       MUST COME OUT TO 7 BLOCKS
;

; Now follows the 1 block of mount table/volume entry information

          .INCLUDE    MWD-HD/MountTable.text   ; dummy mount table/volume entries

       .END
```

```
; File: writemacbblks.Text
;_____________________________________________________________
;
; Hard Disk Driver Write boot blocks routine
;
;    written by Rich Castro
;
;  This program reads Mac boot blocks from internal diskette and then writes
;  them out to the hard disk.
;
;  Modification History:
;  29 Jun 84  RDC  Write initial version
;_____________________________________________________________

          .NOLIST
          .INCLUDE TLASM-SYSEQU.TEXT
          .INCLUDE TLASM-SYSMACS.TEXT
          .INCLUDE TLASM-SYSERR.TEXT
          .LIST


IntrnlDrv .EQU     1                  ; Sony internal drive

          .FUNC    WrtBootBlks,0

;  on entry 4(SP)=result
;  first create space on stack for buffer and parameter block

          MOVEQ    #127,D0
@0        CLR.L    -(SP)              ; clear a buffer area on the stack
          DBRA     D0,@0
          MOVE.L   SP,A1              ; buffer for boot block data

          MOVEQ    #(IOVQElSize/2)-1,D0
@1        CLR.W    -(SP)              ; clear a parameter block on the stack
          DBRA     D0,@1
          MOVE.L   SP,A0

          MOVE.L   A1,IOBuffer(A0)    ; set data buffer ptr
          MOVE.W   #1,IOPosMode(A0)   ; use position mode 1 (from disk start)
          MOVEQ    #0,D0              ; read block 0
          BSR.S    ReadDsk            ; from internal Sony
          BNE.S    WrtXit             ; exit if errors
          MOVEQ    #0,D0              ; else write block 0
          BSR.S    WrtDsk             ;  to hard disk
          BNE.S    WrtXit             ; exit if errors

          MOVE.L   #$200,D0           ; read block 1
          BSR.S    ReadDsk            ; from internal Sony
          BNE.S    WrtXit             ; exit if errors
          MOVE.L   #$200,D0           ; else write block 1
          BSR.S    WrtDsk             ;  to hard disk

WrtXit
          ADD      #(512+IOVQElSize),SP  ; clean up stack space . . .
          MOVE.L   (SP)+,A0           ; get ret addr
```

```
        MOVE    D0,(SP)             ; save result
        JMP     (A0)                ; and return

;-------------------------------------------------------------------------
;  subroutine to read diskette block
;-------------------------------------------------------------------------

ReadDsk MOVE.L  #$200,IOByteCount(A0) ; read one block
        MOVE.L  D0,IOPosOffset(A0)    ; starting block
        MOVE.W  #DskRfn,IORefNum(A0)  ; read from internal drive
        MOVE.W  #IntrnlDrv,IODrvNum(A0)
        _Read
        TST     D0                    ; set return code
        RTS

;-------------------------------------------------------------------------
;  subroutine to write block to hard disk
;-------------------------------------------------------------------------

WrtDsk  MOVE.L  #$200,IOByteCount(A0) ; write one block
        MOVE.L  D0,IOPosOffset(A0)    ; starting block
        MOVE.W  #HDRfNum,IORefNum(A0) ; write to hard disk
        MOVE.W  #HDDrive,IODrvNum(A0)
        _Write
        TST     D0                    ; set return code
        RTS


        .END
```

```
; File: WrtBBlks.Text
;_____________________________________________________________
;
; Hard Disk Driver Write boot blocks routine
;
;    written by Rich Castro
;
; This program reads boot blocks from internal diskette and then writes
; them out to the hard disk.
;
; Modification History:
; 29 Jun 84  RDC  Write initial version
;_____________________________________________________________

         .NOLIST
         .INCLUDE TLASM-SYSEQU.TEXT
         .INCLUDE TLASM-SYSMACS.TEXT
         .INCLUDE TLASM-SYSERR.TEXT
         .LIST


IntrnlDrv .EQU     1                 ; Sony internal drive

         .FUNC     WrtBootBlks,0

; on entry 4(SP)=result
; first create space on stack for buffer and parameter block

         MOVEQ   #127,D0
@0       CLR.L   -(SP)               ; clear a buffer area on the stack
         DBRA    D0,@0
         MOVE.L  SP,A1               ; buffer for boot block data

         MOVEQ   #(IOVQElSize/2)-1,D0
@1       CLR.W   -(SP)               ; clear a parameter block on the stack
         DBRA    D0,@1
         MOVE.L  SP,A0

         MOVE.L  A1,IOBuffer(A0)     ; set data buffer ptr
         MOVE.W  #1,IOPosMode(A0)    ; use position mode 1 (from disk start)
         MOVEQ   #0,D0               ; read block 0
         BSR.S   ReadDsk             ; from internal Sony
         BNE.S   WrtXit              ; exit if errors
         MOVEQ   #0,D0               ; else write block 0
         BSR.S   WrtDsk              ;  to hard disk
         BNE.S   WrtXit              ; exit if errors

         MOVE.L  #$200,D0            ; read block 1
         BSR.S   ReadDsk             ; from internal Sony
         BNE.S   WrtXit              ; exit if errors
         MOVE.L  #$200,D0            ; else write block 1
         BSR.S   WrtDsk              ;  to hard disk

WrtXit
         ADD     #(512+IOVQElSize),SP ; clean up stack space . . .
         MOVE.L  (SP)+,A0            ; get ret addr
```

```
         MOVE    D0,(SP)              ; save result
         JMP     (A0)                 ; and return

;-------------------------------------------------------------------------
;  subroutine to read diskette block
;-------------------------------------------------------------------------

ReadDsk  MOVE.L  #$200,IOByteCount(A0) ; read one block
         MOVE.L  D0,IOPosOffset(A0)    ; starting block
         MOVE.W  #DskRfn,IORefNum(A0)  ; read from internal drive
         MOVE.W  #IntrnlDrv,IODrvNum(A0)
         _Read
         TST     D0                   ; set return code
         RTS

;-------------------------------------------------------------------------
;  subroutine to write block to hard disk
;-------------------------------------------------------------------------

WrtDsk   MOVE.L  #$200,IOByteCount(A0) ; write one block
         MOVE.L  D0,IOPosOffset(A0)    ; starting block
         MOVE.W  #HDRfNum,IORefNum(A0) ; write to hard disk
         MOVE.W  #HDDrive,IODrvNum(A0)
         _Write
         TST     D0                   ; set return code
         RTS


         .END
```